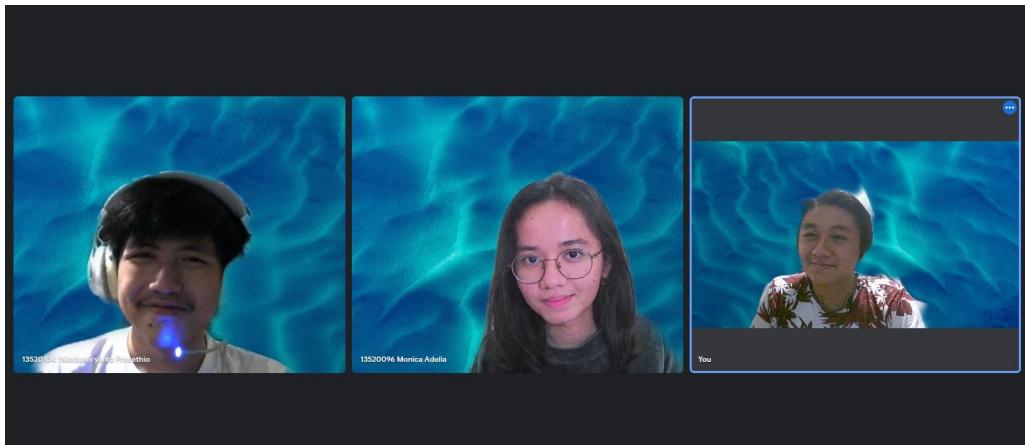


# LAPORAN TUGAS BESAR II

## IF2211 Strategi Algoritma

### Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder Crawling*



Dipersiapkan oleh:  
Kelompok 25 - Finding Nemu

|                           |          |
|---------------------------|----------|
| Monica Adelia             | 13520096 |
| Ilham Bintang Nurmansyah  | 13520102 |
| Yakobus Iryanto Prasethio | 13520104 |

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung  
Jl. Ganesha 10, Bandung 40132

## Daftar Isi

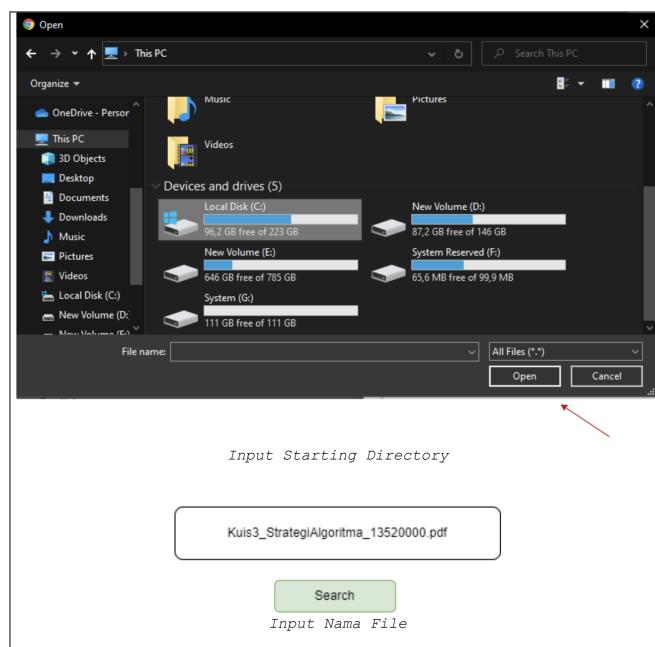
|  |           |
|--|-----------|
| <b>Daftar Isi</b>                                      | <b>2</b>  |
| <b>BAB I</b>   | <b>3</b>  |
| <b>BAB II</b>  | <b>5</b>  |
| Graph Traversal  | 5         |
| Tree   | 5         |
| BFS  | 6         |
| DFS  | 7         |
| C# Desktop Application Development                     | 7         |
| <b>BAB III</b>   | <b>9</b>  |
| Pemecahan Masalah                                      | 9         |
| Mapping Persoalan menjadi Elemen Algoritma BFS dan DFS | 9         |
| Ilustrasi Kasus Lain                                   | 9         |
| <b>BAB IV</b>  | <b>12</b> |
| Repositori GitHub                                      | 12        |
| Link Youtube   | 12        |
| Implementasi Pseudocode                                | 12        |
| Struktur Data dan Spesifikasi Program                  | 13        |
| Tata Cara Penggunaan Program                           | 14        |
| Hasil Pengujian  | 19        |
| Antarmuka program                                      | 19        |
| Data uji   | 21        |
| Analisis Desain Solusi                                 | 27        |
| <b>BAB V</b>   | <b>28</b> |
| Kesimpulan   | 28        |
| Saran  | 28        |
| <b>Daftar Pustaka</b>                                  | <b>29</b> |

## BAB I

### Deskripsi Tugas

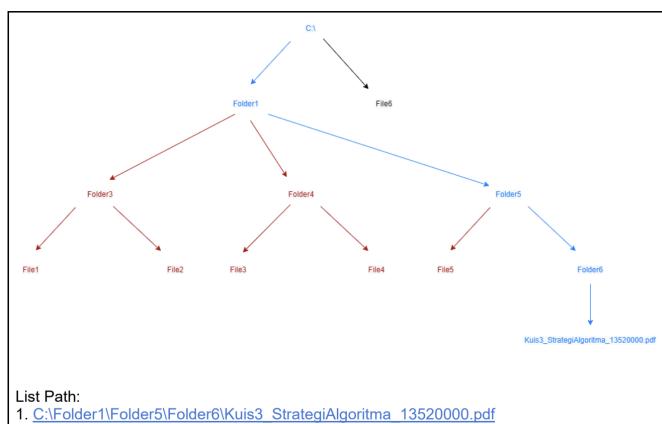
Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari *file explorer* pada sistem operasi, yang pada tugas ini disebut dengan *Folder Crawling*. Dengan memanfaatkan algoritma *Breadth First Search* (BFS) dan *Depth First Search* (DFS),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



Gambar 1 Contoh input program

Contoh *output* aplikasi:

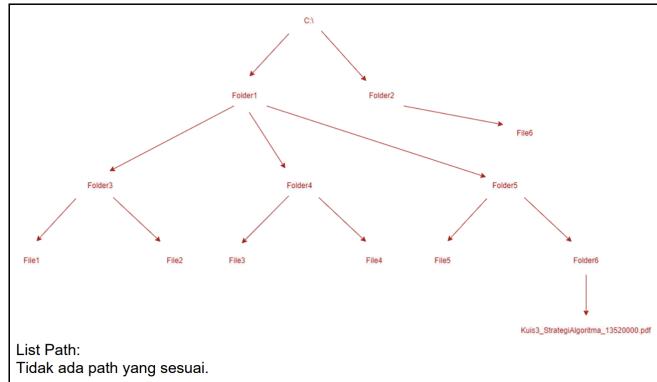


Gambar 2 Contoh output program

Misalkan pengguna ingin mengetahui langkah *folder crawling* untuk menemukan file Kuis3\_StrategiAlgoritma\_13520000.pdf.

Maka, path pencarian DFS adalah sebagai berikut.  $C:\rightarrow \text{Folder1}\rightarrow \text{Folder3}\rightarrow \text{File1}\rightarrow \text{Folder3}\rightarrow \text{File2}\rightarrow \text{Folder3}\rightarrow \text{Folder1}\rightarrow \text{Folder4}\rightarrow \text{File3}\rightarrow \text{Folder4}\rightarrow \text{File4}\rightarrow \text{Folder4}\rightarrow \text{Folder1}\rightarrow \text{Folder5}\rightarrow \text{File5}\rightarrow \text{Folder5}\rightarrow \text{Folder6}\rightarrow \text{Kuis3\_StrategiAlgoritma\_13520000.pdf}$ .

Pada gambar di atas, rute yang dilewati pada pencarian DFS diwarnai dengan warna merah. Sedangkan, rute untuk menuju tempat file berada diberi warna biru. Rute yang masuk antrian tapi belum diperiksa diberi warna hitam. Anda bebas menentukan warnanya asalkan dibedakan antara ketiga hal tersebut.

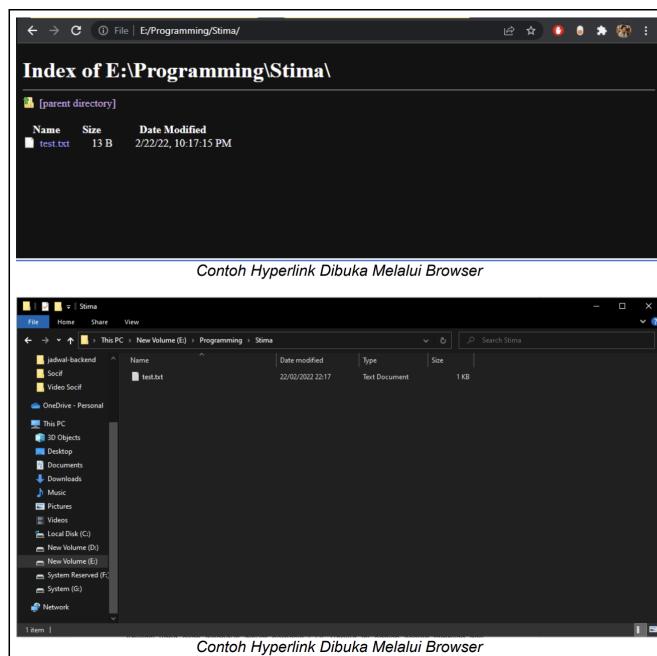


Gambar 3 Contoh output program jika file tidak ditemukan

Jika file yang ingin dicari pengguna tidak ada pada direktori *file*, misalnya saat pengguna mencari Kuis3\_Probstat.pdf, maka path pencarian DFS adalah sebagai berikut:  $C:\rightarrow \text{Folder1}\rightarrow \text{Folder3}\rightarrow \text{File1}\rightarrow \text{Folder3}\rightarrow \text{File2}\rightarrow \text{Folder3}\rightarrow \text{Folder1}\rightarrow \text{Folder4}\rightarrow \text{File3}\rightarrow \text{Folder4}\rightarrow \text{File4}\rightarrow \text{Folder4}\rightarrow \text{Folder1}\rightarrow \text{Folder5}\rightarrow \text{File5}\rightarrow \text{Folder5}\rightarrow \text{Folder6}\rightarrow \text{Kuis3\_StrategiAlgoritma\_13520000.pdf}\rightarrow \text{Folder6}\rightarrow \text{Folder5}\rightarrow \text{Folder1}\rightarrow C:\rightarrow \text{Folder2}\rightarrow \text{File6}$ .

Pada gambar di atas, semua simpul dan cabang berwarna merah yang menandakan seluruh direktori sudah selesai diperiksa semua namun tidak ada yang mengarah ke tempat file berada.

Contoh *hyperlink* pada path:



Gambar 4 Contoh ketika hyperlink di-klik

## BAB II

### Landasan Teori

#### A. Graph Traversal

Graf adalah sekumpulan objek terstruktur dimana beberapa pasangan objek mempunyai hubungan atau keterkaitan tertentu. Graf terdiri dari himpunan objek-objek yang dinamakan titik, simpul, atau sudut, yang dihubungkan oleh penghubung yang disebut garis atau sisi. Traversal adalah proses kunjungan dalam graf atau pohon, dengan setiap node hanya dikunjungi tepat satu kali.

Traversal graf memiliki arti mengunjungi simpul-simpul dengan cara yang sistematis. Dalam traversal graf ini terdapat dua cara, yaitu:

1. Pencarian Melebar (*Breadth First Search / BFS*)
2. Pencarian Mendalam (*Depth First Search / DFS*)

Dalam kedua cara traversal graf ini, digunakan asumsi bahwa tiap simpul/node/titik setidaknya terhubung dengan satu simpul/node/titik lainnya dan graf merupakan graf terhubung. Traversal graf ini termasuk algoritma pencarian solusi, dimana algoritma pencarian solusi ini terbagi menjadi dua jenis, yaitu:

1. Tanpa Informasi (*uninformed / blind search*)

Algoritma pencarian tanpa informasi ini dilakukan tanpa ada informasi tambahan. Contohnya adalah DFS, BFS, *Depth Limited Search*, *Iterative Deepening Search*, *Uniform Cost Search*.

2. Dengan Informasi (*informed search*)

Algoritma pencarian dengan informasi ini merupakan pencarian berbasis heuristik. Pada algoritma ini, diketahui *non-goal state* yang “lebih menjanjikan” daripada yang lain. Contohnya adalah *Best First Search* dan *A\**.

Dalam proses pencarian atau pengunjungan semua node pada graf, terdapat dua pendekatan:

1. Graf Statis

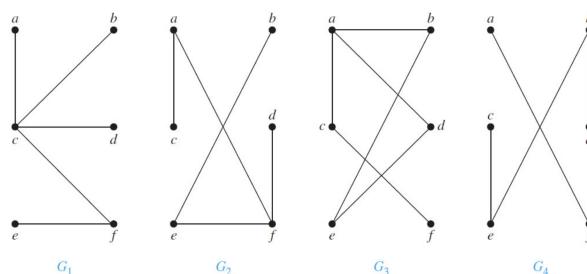
Pada pendekatan graf statis ini, graf sudah diciptakan sebelum proses pencarian dilakukan.

2. Graf Dinamis

Pada pendekatan graf dinamis ini, graf baru dibuat saat proses pencarian sedang dilakukan. Sebelum pencarian, tidak tersedia graf.

#### B. Tree

Pohon adalah graf tak-berarah yang terhubung, namun tidak mengandung sirkuit. Sebuah pohon tidak akan mempunyai cabang ganda ataupun cabang yang melingkar.



Gambar 5 Contoh pohon dan yang bukan pohon  
(sumber: <https://www.haimatematika.com/2018/12/graf-pohon-teori-graf.html>)

Graf G1 dan Graf G2 merupakan graf pohon karena tidak terdapat sisi ganda dan sirkuit. Graf G3 memuat sirkuit sehingga tidak termasuk graf pohon. Graf G4 bukan graf pohon karena tidak terhubung.

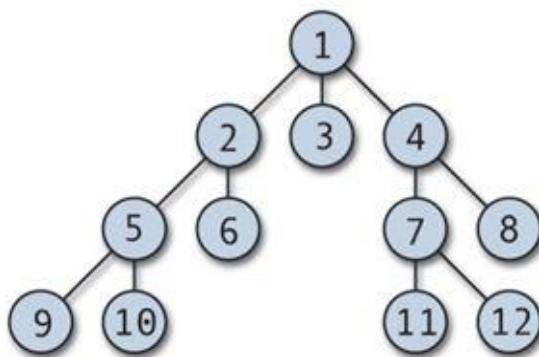
Salah satu contoh penerapan dari graf pohon ini adalah *folder* penyimpanan pada laptop/komputer/alat elektronik kita. Setiap *folder* dapat terdiri dari *file* maupun *folder* lagi. Setiap *folder/file* pada suatu penyimpanan pasti terhubung satu sama lain pada suatu *base folder*. *Folder* dan *File* pada penyimpanan juga tidak mengandung sirkuit. Oleh karena itu, folder penyimpanan dapat direpresentasikan dengan graf pohon.

### C. BFS

Algoritma BFS (*Breadth First Search*) adalah salah satu algoritma yang digunakan untuk pencarian jalur. Algoritma ini adalah salah satu algoritma pencarian jalur sederhana, dimana pencarian dimulai dari simpul awal, kemudian dilanjutkan ke semua simpul yang bertetangga dengan simpul awal secara terurut. Jika simpul tujuan belum ditemukan, maka perhitungan akan diulang lagi ke masing-masing simpul cabang dari masing-masing simpul, sampai simpul tujuan ditemukan.

Breadth First Search (BFS) adalah metode traversing yang digunakan dalam grafik. Ini menggunakan antrian untuk menyimpan simpul yang dikunjungi. Dalam metode ini yang ditekankan adalah pada simpul grafik, satu simpul dipilih pada awalnya kemudian dikunjungi dan ditandai. Simpul yang berdekatan dengan simpul yang dikunjungi kemudian dikunjungi dan disimpan dalam antrian berurutan. Demikian pula, simpul yang disimpan kemudian diperlakukan satu per satu, dan simpul yang berdekatan dikunjungi. Sebuah node sepenuhnya dieksplorasi sebelum mengunjungi node lain dalam grafik, dengan kata lain, node tersebut melintasi node yang belum dieksplorasi paling dangkal terlebih dahulu.

Berikut ini adalah urutan pengunjungan suatu node pada *graph* (pohon) menggunakan algoritma BFS:



Gambar 6 Contoh BFS pada pohon  
(sumber: <https://onbubble.wordpress.com/2011/05/26/6/>)

Dalam algoritma BFS, simpul anak yang telah dikunjungi disimpan dalam suatu antrian. Antrian ini digunakan untuk mengacu simpul-simpul yang bertetangga dengannya yang akan dikunjungi kemudian sesuai urutan pengantrean. Berikut ini langkah-langkah algoritma BFS:

1. Masukkan simpul ujung/akar/root ke dalam antrian.
2. Ambil simpul dari awal antrian, lalu cek apakah simpul merupakan solusi.
3. Jika simpul merupakan solusi, pencarian selesai dan hasil dikembalikan.

4. Jika simpul bukan solusi, seluruh simpul yang bertetangga dengan simpul tersebut. (simpul anak) masuk ke dalam antrian secara berurutan.
5. Jika antrian kosong dan setiap simpul sudah dicek, pencarian selesai dan mengembalikan hasil solusi tidak ditemukan.
6. Ulangi pencarian dari langkah kedua.

#### D. DFS

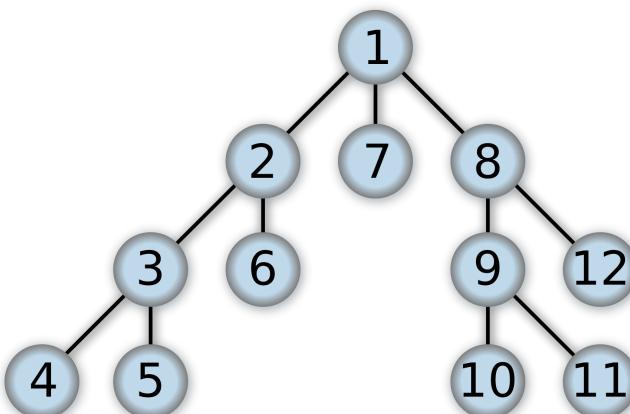
DFS (Depth First Search) adalah salah satu algoritma penelusuran struktur graf / pohon berdasarkan kedalaman. Simpul ditelusuri dari root kemudian ke salah satu simpul anaknya (misalnya prioritas penelusuran berdasarkan anak pertama [simpul sebelah kiri] ), maka penelusuran dilakukan terus melalui simpul anak pertama dari simpul anak pertama level sebelumnya hingga mencapai level terdalam.

Setelah sampai di level terdalam, penelusuran akan kembali ke 1 level sebelumnya untuk menelusuri simpul anak kedua pada pohon biner [simpul sebelah kanan] lalu kembali ke langkah sebelumnya dengan menelusuri simpul anak pertama lagi sampai level terdalam dan seterusnya.

Dalam implementasinya DFS dapat diselesaikan dengan cara rekursif atau dengan bantuan struktur data stack. Langkah-langkah DFS dengan metode rekursif adalah sebagai berikut:

1. Kunjungi simpul v (simpul awal / root)
2. Kunjungi simpul w yang bertetangga dengan simpul v
3. Ulangi DFS langkah 1 dengan w sebagai simpul awal
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian diruntut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dari simpul yang telah dikunjungi.

Berikut ini adalah urutan pengunjungan suatu node pada *graph* (pohon) menggunakan algoritma DFS:



Gambar 7 Contoh DFS pada pohon

(sumber: [https://en.wikipedia.org/wiki/Depth-first\\_search#/media/File:Depth-first-tree.svg](https://en.wikipedia.org/wiki/Depth-first_search#/media/File:Depth-first-tree.svg) )

#### E. C# Desktop Application Development

C# merupakan salah satu bahasa pemrograman berorientasi objek. C# biasa digunakan untuk membangun aplikasi desktop ataupun *mobile*, *server-side website*, dan lain-lain. Seperti bahasa pemrograman berorientasi objek lainnya, C# menerapkan konsep-konsep objek.

C# Desktop Application Development adalah salah satu dari banyak cara untuk membuat

aplikasi desktop. Ada beberapa framework untuk membuat Desktop Application, salah satunya adalah menggunakan .NET Winforms. Framework .NET Winforms sendiri memungkinkan kita untuk membuat Desktop Application di sistem operasi Windows. IDE yang digunakan untuk membuat C# Desktop Application Development ini salah satunya adalah Microsoft Visual Studio (berbeda dengan Visual Studio Code).

Pada Tugas Besar kali ini digunakan Window Form Application untuk merealisasikan GUI (*Graphical User Interface*) dari program yang dibuat. Dengan memanfaatkan Visual Studio, langkah-langkah untuk membuatnya adalah sebagai berikut:

1. Unduh Visual Studio
2. *Install* dan jalankan Visual Studio
3. Pada halaman utama, pilih “Create a new project”
4. Pilih Windows Forms App sebagai template dari aplikasi yang ingin dibuat
5. Tuliskan nama dari aplikasi dan pilih lokasi penyimpanan dari aplikasi yang Anda buat pada window yang muncul
6. Pilih jenis *Framework* yang ingin digunakan
7. Ketik “Create”
8. Lalu akan muncul *default* form dari Windows Form.
9. Untuk menjalankan aplikasi, klik tombol panah ke kanan berwarna hijau. Visual studio akan melakukan *build* terhadap aplikasi dan jika berhasil, .exe akan secara otomatis dijalankan.

## BAB III

### Analisis Pemecahan Masalah

#### A. Pemecahan Masalah

Dari permasalahan yang disebutkan pada Bab 1, telah dilakukan beberapa langkah-langkah pemecahan masalah antara lain:

1. Memahami permasalahan yang diberikan pada Bab 1
2. Memahami konsep algoritma BFS dan DFS dalam traversal pohon direktori
3. Membagi permasalahan menjadi elemen BFS dan DFS
4. Melakukan testing terhadap algoritma BFS dan DFS yang telah didesain
5. Mempelajari bahasa pemrograman C# dan melakukan instalasi Visual Studio sebagai IDE yang akan digunakan untuk melakukan pemrograman dengan bahasa C#
6. Membuat GUI dalam Windows Forms Application yang dapat menerima masukkan pengguna berupa root folder, nama file yang dicari, pilihan algoritma (BFS atau DFS), dan pilihan untuk menemukan semua kemunculan file.
7. Membuat visualisasi pohon pada GUI berdasarkan masukkan root folder dan pencarian BFS atau DFS dengan library MSAGL
8. Melakukan testing terhadap GUI yang telah dibuat

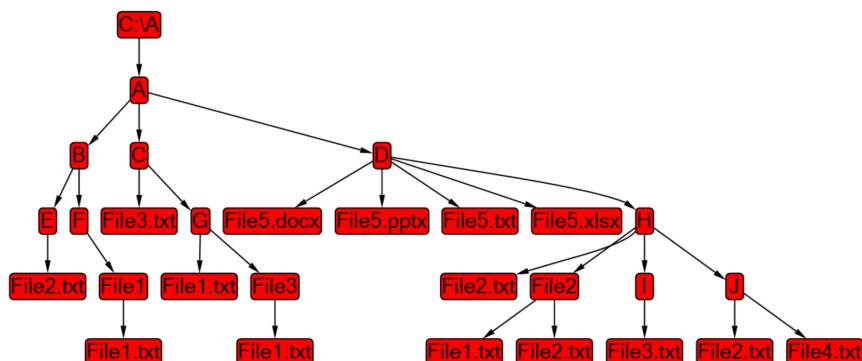
#### B. Mapping Persoalan menjadi Elemen Algoritma BFS dan DFS

Misalkan nama file yang dicari adalah X, dengan file/folder sebagai simpul dan direktori/path sebagai sisi.

- a. Problem state: Mencari direktori/path file pada pohon yang namanya sesuai dengan nama file
- b. Initial state: Root folder
- c. Solution state: File yang memiliki nama sama seperti file yang dicari
- d. State Space: Himpunan semua folder yang ada pada pohon

#### C. Ilustrasi Kasus Lain

Contoh pencarian “File9.txt” dari sebuah pohon folder menggunakan Algoritma BFS dengan All Occurrence tidak dipilih



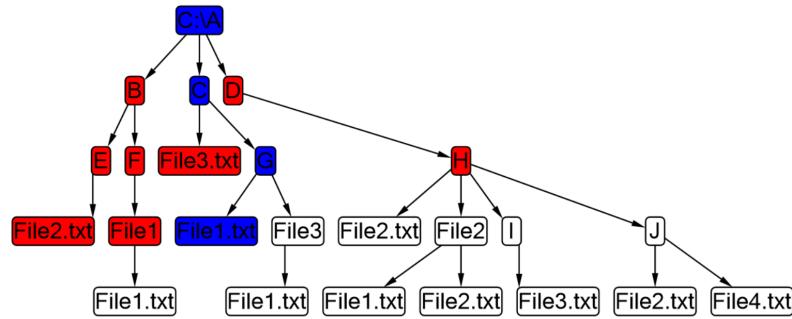
Gambar 8 Pencarian file File9.txt menggunakan BFS, tidak All Occurrence

Urutan pencarian menggunakan algoritma BFS adalah:

C:\A → B → C → D → E → F → File3.txt → G → H → File2.txt → File1 → File1.txt → File3 → File2.txt → File2 → I → J → File1.txt → File1.txt → File2.txt → File3.txt → File2.txt → File4.txt.

File tidak ditemukan karena tidak ada file bernama File9.txt

Contoh pencarian “File1.txt” dari sebuah pohon folder menggunakan Algoritma BFS dengan All Occurrence tidak dipilih

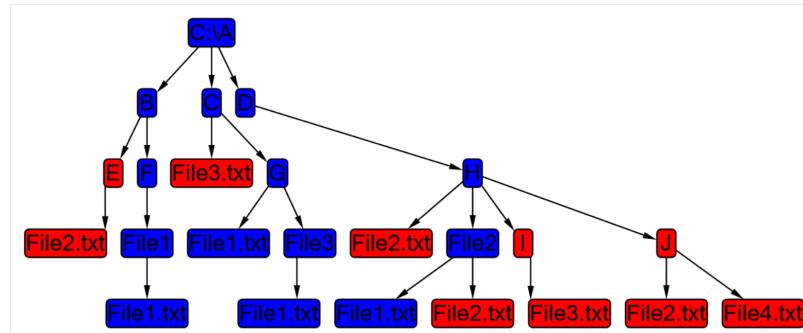


Gambar 9 Pencarian file File1.txt menggunakan BFS, tidak All Occurrence

Urutan pencarian menggunakan algoritma BFS adalah:

C:\A → B → C → D → E → F → File3.txt → G → H → File2.txt → File1 → **File1.txt**

Contoh pencarian “File1.txt” dari sebuah pohon folder menggunakan Algoritma BFS dengan All Occurrence dipilih

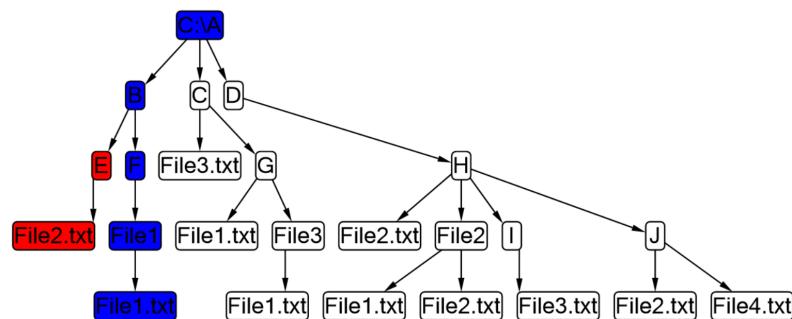


Gambar 10 Pencarian file File1.txt menggunakan BFS, All Occurrence

Urutan pencarian menggunakan algoritma BFS adalah:

C:\A → B → C → D → E → F → File3.txt → G → H → File2.txt → File1 → **File1.txt** → File3 → File2.txt → File2 → I → J → **File1.txt** → **File1.txt** → File1.txt → File2.txt → File3.txt → File2.txt → File4.txt

Contoh pencarian “File1.txt” dari sebuah pohon folder menggunakan Algoritma DFS dengan All Occurrence tidak dipilih

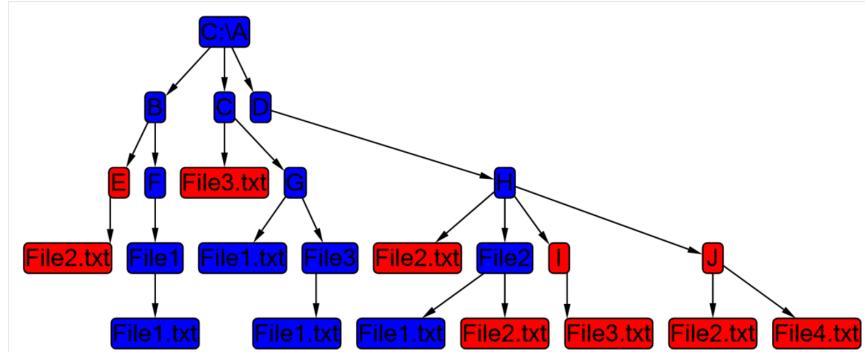


Gambar 11 Pencarian file File1.txt menggunakan DFS, tidak All Occurrence

Urutan pencarian menggunakan algoritma DFS adalah:

C:\A → B → E → File2.txt → F → File1 → **File1.txt**

Contoh pencarian “File1.txt” dari sebuah pohon folder menggunakan Algoritma DFS dengan All Occurrence dipilih

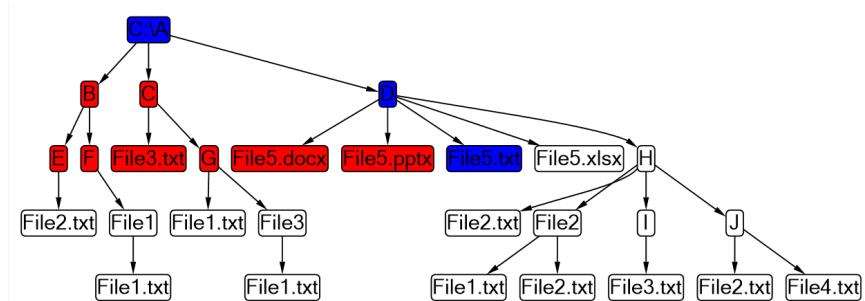


Gambar 12 Pencarian file File1.txt menggunakan DFS, All Occurrence

Urutan pencarian menggunakan algoritma DFS adalah:

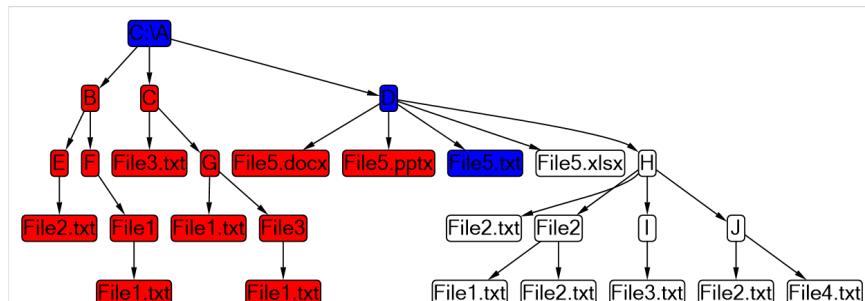
C:\A → B → E → File2.txt → F → File1 → **File1.txt** → C → File3.txt → G → **File1.txt** → File3 → **File1.txt** → D → H → File2.txt → File2 → **File1.txt** → File2.txt → I → File3.txt → J → File2.txt → File4.txt

Contoh pencarian “File5.txt” dari sebuah pohon folder menggunakan Algoritma BFS dengan All Occurrence tidak dipilih



Gambar 13 Pencarian file File5.txt menggunakan BFS, tidak All Occurrence

Contoh pencarian “File5.txt” dari sebuah pohon folder menggunakan Algoritma DFS dengan All Occurrence tidak dipilih



Gambar 14 Pencarian file File5.txt menggunakan DFS, tidak All Occurrence

Dapat dilihat bahwa di dalam folder D terdapat 4 file dengan nama yang sama tetapi ekstensi yang berbeda (docx, pptx, txt, dan xlsx). Algoritma BFS dan DFS tetap mencari file yang memiliki nama dan ekstensi yang sama.

## BAB IV

### Implementasi dan Pengujian

#### A. Repotori GitHub

<https://github.com/YakobusIP/Tubes2-STIMA.git>

#### B. Link Youtube

<https://bit.ly/VideoYoutubeFindingNemu>

#### C. Implementasi Pseudocode

Pseudocode sebagai berikut merupakan bagian dari implementasi main program *searching* dalam *folder crawling*. Untuk komentar dimulai dengan “//” untuk memperjelas code dan bukan merupakan bagian dari pseudocode.

```
//BFS Search
BFSSearch(input string filename, input TreeNode root, input Boolean findAll)
KAMUS LOKAL
    strQ : Queue of TreeNode
    haveVisited : List of string
    path : List of string
    wayToPath : List of String
    cekNamaFile : string

ALGORITMA
    strQ.enqueue(root)

    while (strQ.Count > 0) do
        cekNamaFile <- file name dari elemen pertama queue strQ

        if (jumlah children treenode pertama queue = 0)then:
            if (cekNamaFile = filename)then:
                path.add_element(directory cekNamaFile)

                if(findAll = false)then: {find 1}
                    {find 1}
                    hapus semua elemen di queue strQ
                else:
                    {find all}
                    strQ.dequeue
            else:
                haveVisited.add_element(directory strQ.first)
                strQ.dequeue
        else:
            for (children in TreeNode pertama queue) do:
                strQ.enqueue(children)

                haveVisited.add_element(directory strQ.first)
                dequeue(strQ)

            if (isi list path != 0)then:
                wayToPath <- BreakPath(path, root.getFolderName())

    -> path, haveVisited, wayToPath

// DFS Search
DFSSearch(input string folderName, input TreeNode root, input List<string> path, input
List<string> visitedDirectory, input boolean findAll) {

KAMUS LOKAL
    rootFullDirectory : string
    child: TreeNode
```

```

path : List of string
visitedDirectory : List of String

ALGORITMA
if (findAll = true) then
    { Base of recursion }
    string rootFullDirectory <- root.getFolderName()
    if (Path.GetFileName(rootFullDirectory) = folderName) then
        path.Add(rootFullDirectory)
        -> path, visitedDirectory

    // Add root to visitedDirectory
    if (visitedDirectory.Count() = 0) then
        visitedDirectory.Add(rootFullDirectory)

foreach (var child in root.getChildren())
{ If the dir has been checked thus added to visitedDirectory, then skip it }
    if (!visitedDirectory.Contains(child.getFolderName())) then
        { Add to the list of visitedDirectory and recurse the function }
        visitedDirectory.Add(child.getFolderName())
        DFSsearch(folderName, child, path, visitedDirectory, findAll)

        -> path, visitedDirectory
else
    { Base of recursion }
    string rootFullDirectory <- root.getFolderName()
    if (Path.GetFileName(rootFullDirectory) = folderName) then
        path.Add(rootFullDirectory)
        -> path, visitedDirectory

    { Add root to visitedDirectory }
    if (visitedDirectory.Count() = 0) then
        visitedDirectory.Add(rootFullDirectory)

foreach (child in root.getChildren())
{ If the dir has been checked thus added to visitedDirectory, then skip it }
{ Recurse only if the file hasn't been found yet }
    if ((not visitedDirectory.Contains(child)) and (path.Count() = 0)) then
        { Add to the list of visitedDirectory and recurse the function }
        visitedDirectory.Add(child.getFolderName())
        DFSsearch(folderName, child, path, visitedDirectory, findAll)

    -> path, visitedDirectory

```

## D. Struktur Data dan Spesifikasi Program

|  |
|--|
| class TreeNode   |
| <ul style="list-style-type: none"> <li>- folderName: string</li> <li>- children: List&lt;TreeNode&gt;</li> </ul>   |
| <ul style="list-style-type: none"> <li>+ TreeNode(string)</li> <li>+ AddChild(string): void</li> <li>+ AddChildTree(TreeNode): void</li> <li>+ getChildren(): List&lt;TreeNode&gt;</li> <li>+ getFolderName(): string</li> </ul> |

## E. Tata Cara Penggunaan Program

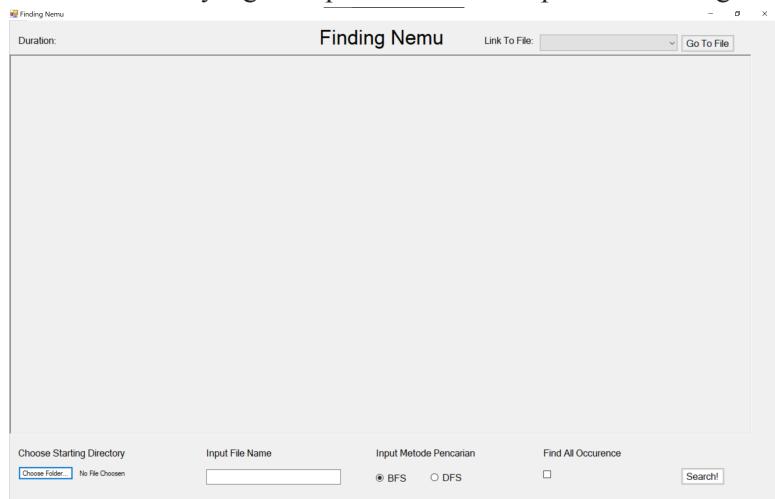
Cara menggunakan program Finding Nemu adalah sebagai berikut:

1. Jalankan Aplikasi Tubes 2 Stima yang ada di folder bin.

| Name                                | Date modified    | Type                 | Size     |
|-------------------------------------|------------------|----------------------|----------|
| AutomaticGraphLayout.dll            | 17/03/2022 0:17  | Application exten... | 1.565 KB |
| AutomaticGraphLayout.Drawing.dll    | 17/03/2022 0:17  | Application exten... | 148 KB   |
| Microsoft.Msagl.GraphViewerGdi.dll  | 17/03/2022 0:17  | Application exten... | 153 KB   |
| Microsoft.Msagl.WpfGraphControl.dll | 17/03/2022 0:17  | Application exten... | 66 KB    |
| Tubes 2 Stima                       | 23/03/2022 19:23 | Application          | 21 KB    |
| Tubes 2 Stima.exe                   | 13/03/2022 15:57 | Configuration Sou... | 1 KB     |
| Tubes 2 Stima.pdb                   | 23/03/2022 19:23 | Program Debug D...   | 56 KB    |

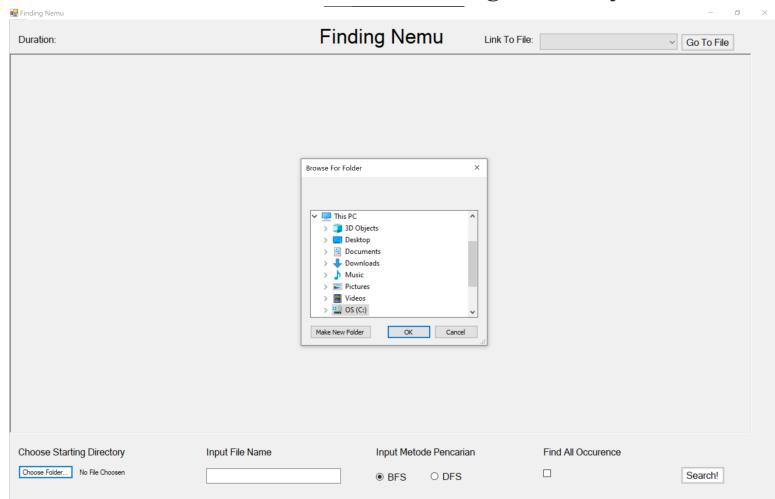
Gambar 15 Aplikasi Tubes 2 Stima pada folder bin

2. Akan muncul window baru yang merupakan menu dari aplikasi “Finding Nemu”



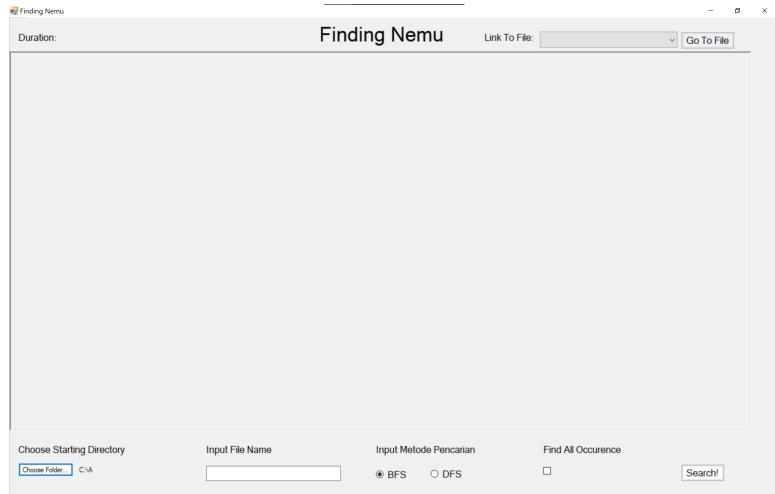
Gambar 16 Tampilan awal aplikasi Finding Nemu

3. Klik tombol “Choose Folder” untuk memilih Starting Directory.



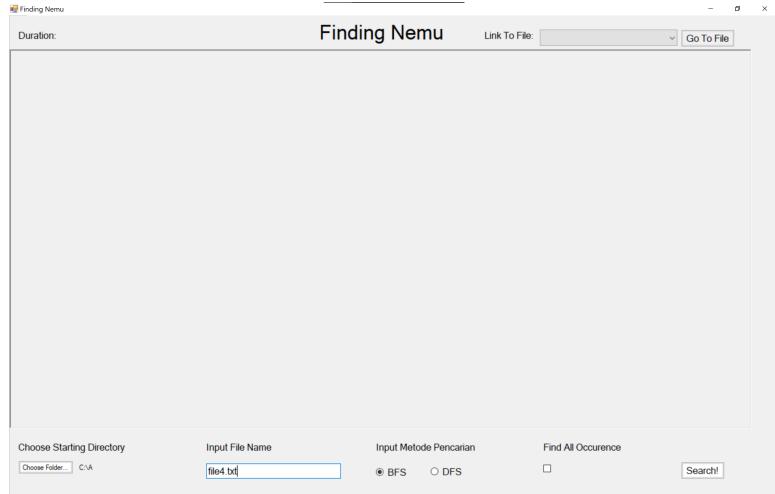
Gambar 17 Tampilan aplikasi untuk memilih starting/root directory

4. Setelah memilih Starting Directory, klik “OK”. Tampilan aplikasi akan seperti dibawah ini.



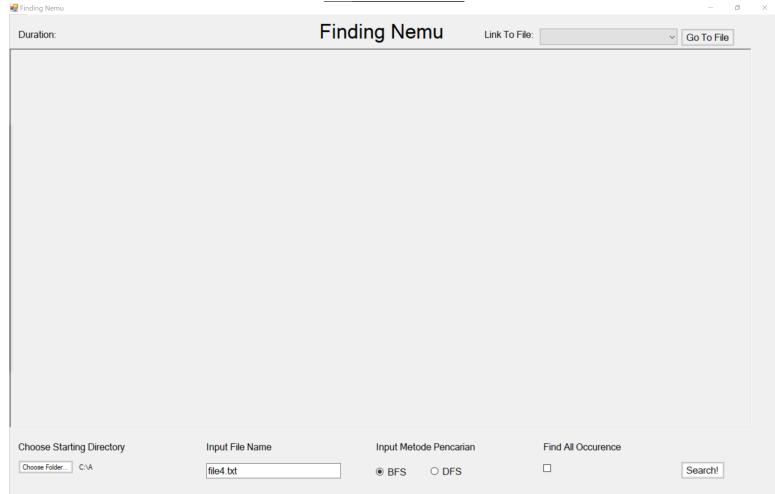
Gambar 18 Tampilan aplikasi setelah starting/root folder dipilih

5. Masukkan nama file yang ingin dicari pada proses folder crawling.

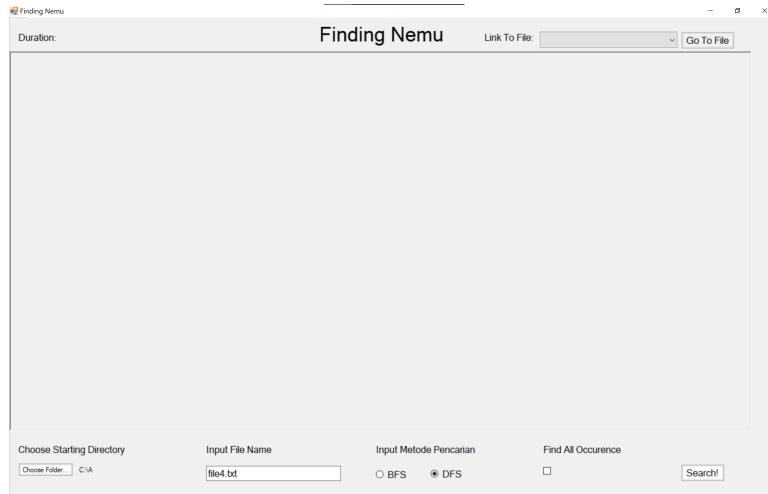


Gambar 19 Tampilan aplikasi ketika pengisian nama file yang ingin dicari

6. Pada bagian Input Metode Pencarian, pilih metode pencarian apa yang ingin digunakan untuk mencari file yang ingin dicari, menggunakan *Breadth First Search* (BFS) atau *Depth First Search* (DFS).

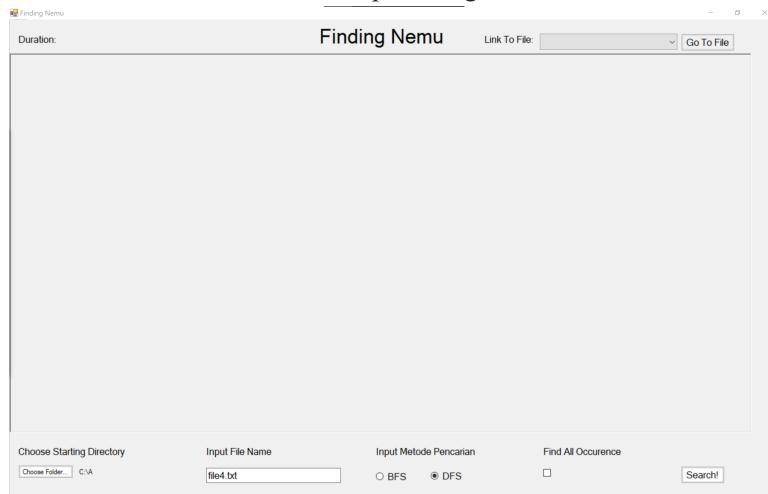


Gambar 20 Tampilan aplikasi ketika memilih metode BFS

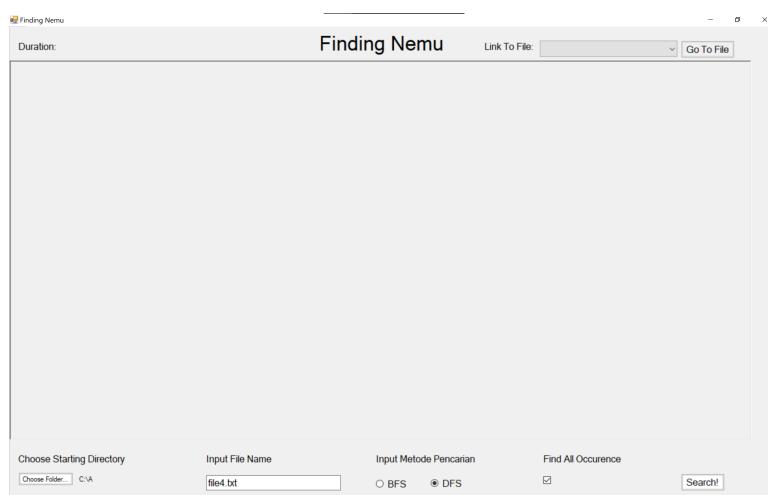


Gambar 21 Tampilan aplikasi ketika memilih metode DFS

7. Berikutnya, pada bagian Find All Occurrence, centang box dengan cara mengklik kotak hingga muncul centang jika ingin mencari semua kemunculan file pada *folder root* atau *starting folder*. Jika tidak, biarkan box tetap kosong.

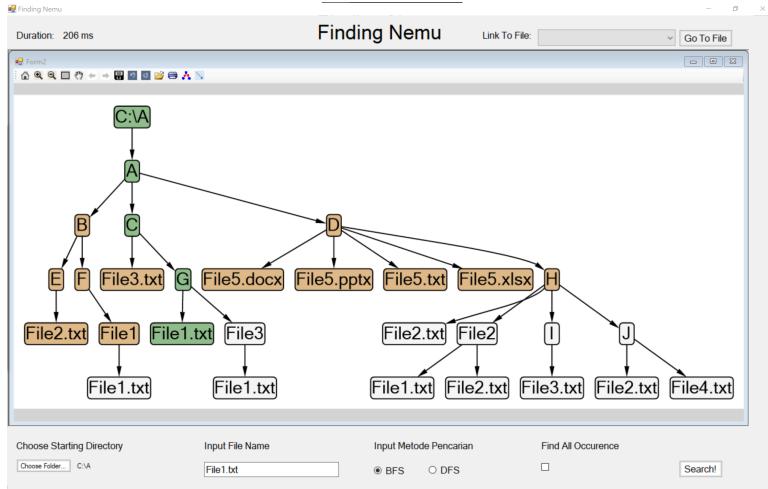


Gambar 22 Tampilan aplikasi ketika tidak memilih All Occurrence

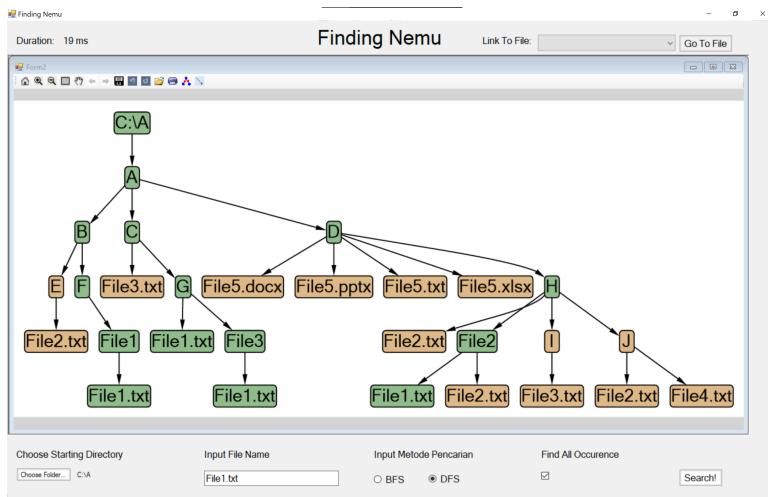


Gambar 23 Tampilan aplikasi ketika memilih All Occurrence

8. Lalu, klik button Search! dan aplikasi akan menampilkan *graph* atau pohon dari folder beserta animasi warna pada proses pencarian menurut metode pilihan. Bila, file ditemukan, aplikasi juga akan menampilkan file temuan dengan warna berbeda.



Gambar 24 Aplikasi menampilkan graph/pohon hasil pencarian BFS tidak All Occurrence



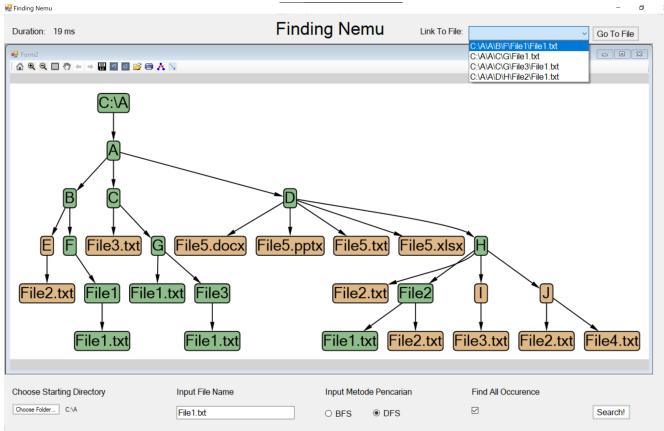
Gambar 25 Aplikasi menampilkan graph/pohon hasil pencarian DFS All Occurrence

9. Pada bagian kiri atas juga dapat dilihat waktu yang dibutuhkan untuk menemukan file yang dicari.



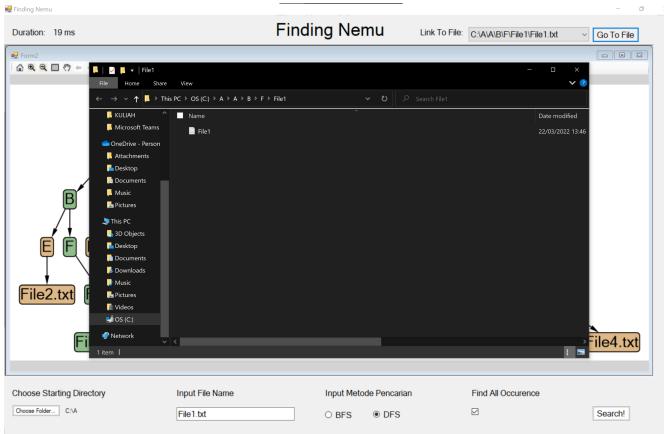
Gambar 26 Tampilan aplikasi menunjukkan durasi pencarian file

10. Pada bagian kanan atas terdapat fitur Link To File yang akan menampilkan directory menuju file inputan yang ingin dicari.



Gambar 27 Tampilan aplikasi menunjukkan directory file yang ditemukan

11. Setelah memilih salah satu directory menuju file, klik button “Go To File”. Akan terbuka window file explorer, dengan directory menuju file tujuan sehingga dapat mengakses file inputan yang dicari.



Gambar 28 Tampilan aplikasi yang menunjukkan lokasi file dengan pop up window

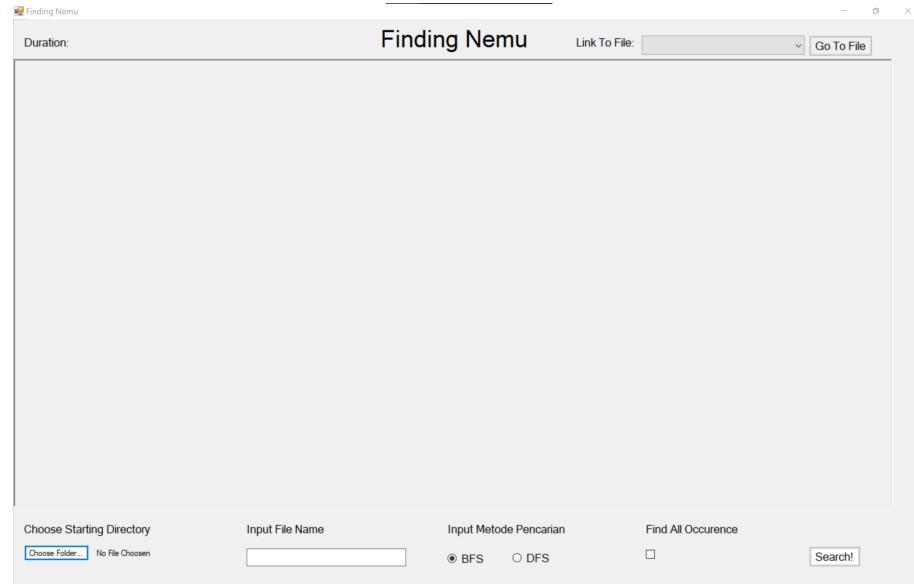
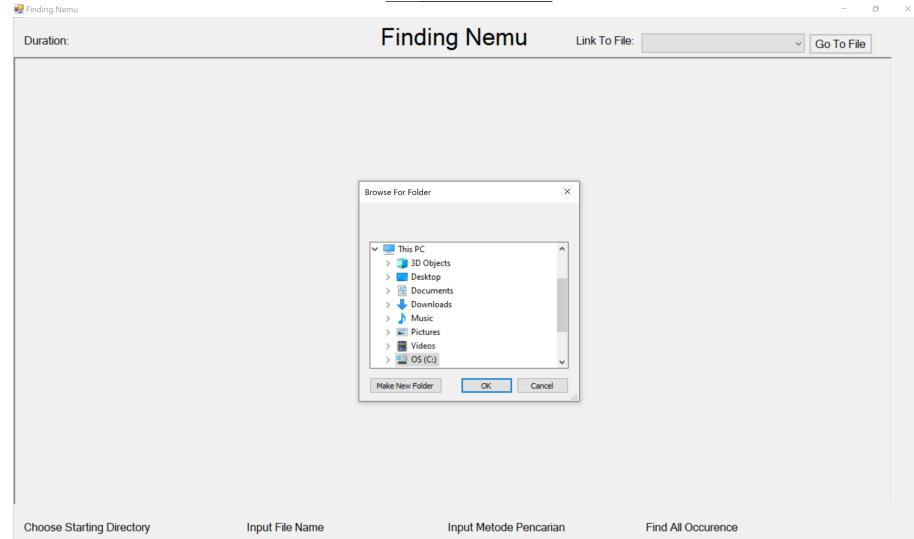
12. Jika sudah selesai menggunakan aplikasi, klik tombol “X” pada pojok kanan atas aplikasi untuk keluar dan menutup aplikasi.

Fitur-fitur yang tersedia pada aplikasi ini adalah pencarian file dari suatu *root directory*. Dimana untuk proses pencarinya, ditawarkan dua jenis pencarian yaitu menggunakan *Breadth First Search* (BFS) dan *Depth First Search* (DFS). Selain itu, aplikasi ini menyediakan fitur untuk pencarian semua kemunculan file pada inputan *root directory* dan bisa juga melakukan pencarian hanya untuk file pertama yang muncul. Aplikasi ini menampilkan *graph* atau pohon dari folder masukan beserta isinya hingga komponen file terkecil. Jika file ditemukan, pohon akan menampilkan urutan folder *directory* menuju file pada tampilan *graph* dengan warna berbeda. Selain itu, dalam penampilan *graph*, aplikasi ini menunjukkan langkah-langkah yang dilakukan algoritma dalam menemukan file yang dicari. Langkah-langkah yang dimaksud disini adalah tiap folder atau file yang dikunjungi dalam mencari file yang ingin dicari. Animasi pengunjungan folder ketika proses pencarian ditunjukkan dalam bentuk perubahan warna pada *graph*. Selain itu, aplikasi juga menunjukkan durasi dalam proses pencarian file ini. Proses ini dituliskan dalam satuan milisecond. Terakhir, aplikasi ini dapat menampilkan directory dari file yang dicari, baik itu hanya satu maupun semua kemunculannya (jika dipilih all occurrence). Dari fitur ini pun bisa langsung menunjukkan keberadaan file tersebut dengan menampilkan window baru file explorer yang langsung menunjukkan file tersebut.

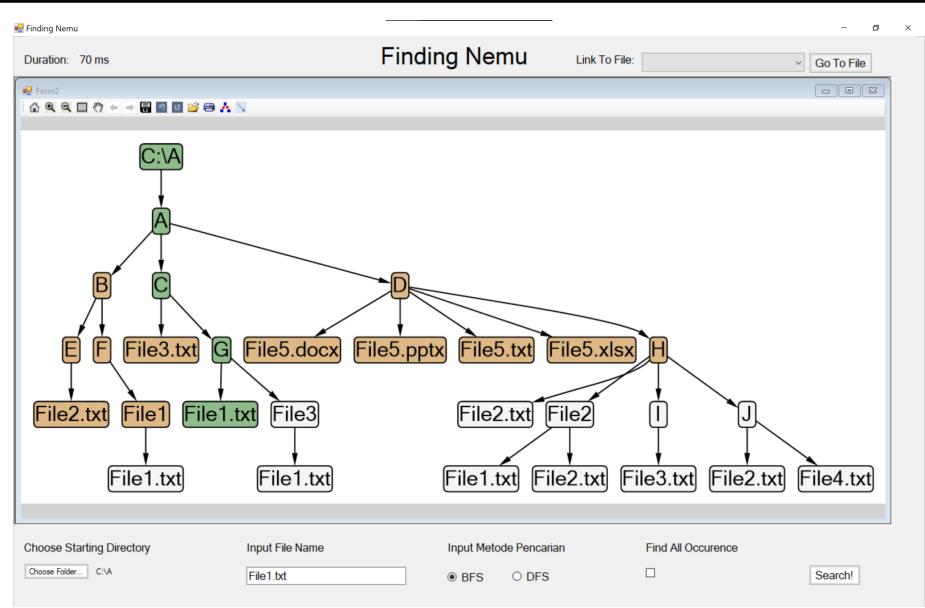
## F. Hasil Pengujian

### a. Antarmuka program

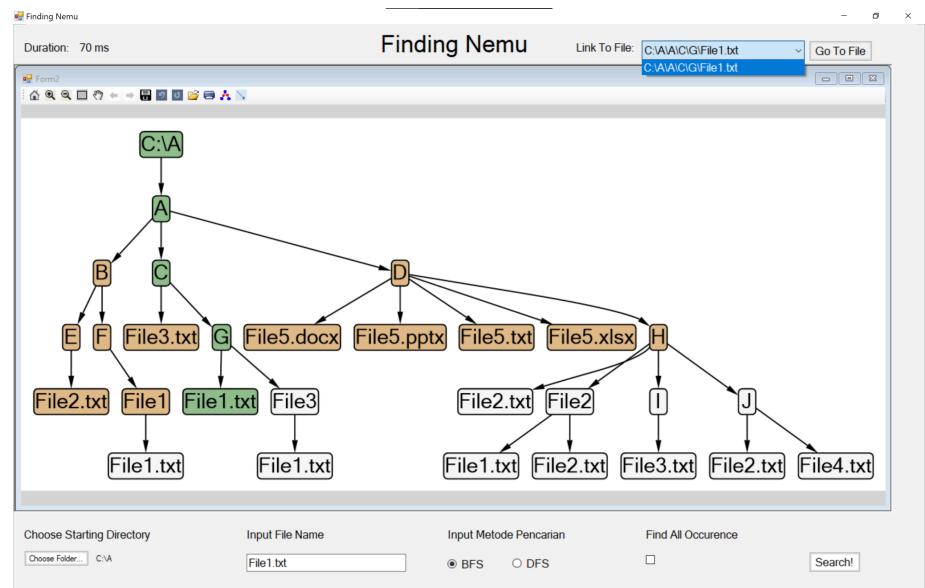
Tabel 1 Antarmuka program

| Antarmuka Program                     |  |
|---------------------------------------|--|
| Tampilan awal                         |    |
| Tampilan pemilihan starting directory |  |

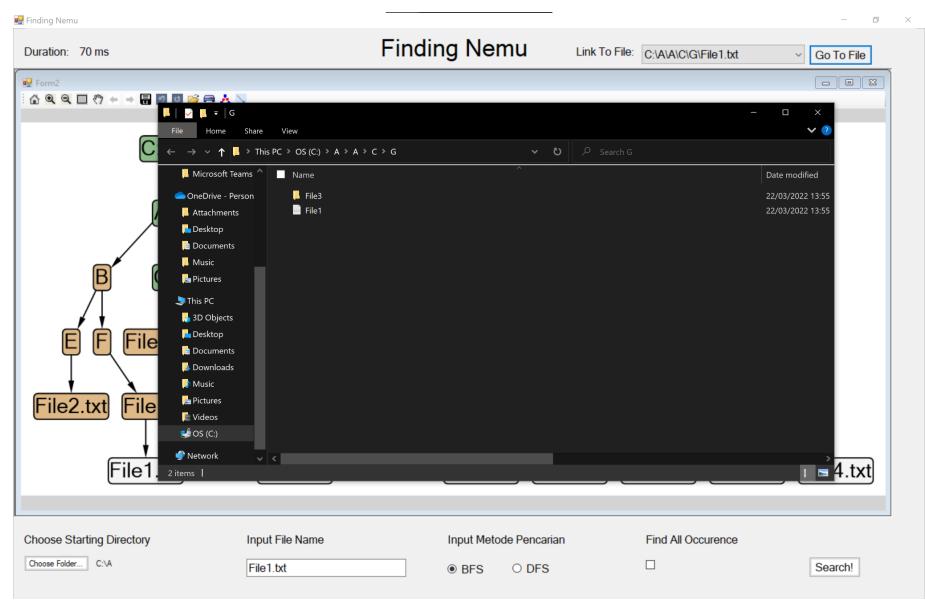
Hasil pencarian



Menu untuk membuka directory hasil pencarian



Tampilan file hasil pencarian



b. Data uji

Tabel 2 Data uji 1

| TESTING 1              |   |
|------------------------|---|
| Skenario Pengujian     | <ul style="list-style-type: none"> <li>• Menggunakan BFS</li> <li>• Tidak All Occurrence</li> <li>• File tidak ditemukan</li> </ul> |
| Root Folder            | C:\A  |
| Nama file yang dicari  | hambin.txt  |
| Hasil Pengujian:       |   |
|                        |   |
| Waktu pencarian: 12 ms |   |

Tabel 3 Data uji 2

| TESTING 2              |   |
|------------------------|---|
| Skenario Pengujian     | <ul style="list-style-type: none"> <li>• Menggunakan BFS</li> <li>• All Occurrence</li> <li>• File tidak ditemukan</li> </ul> |
| Root Folder            | C:\A  |
| Nama file yang dicari  | hambin.txt  |
| Hasil Pengujian:       |   |
|                        |   |
| Waktu pencarian: 13 ms |   |

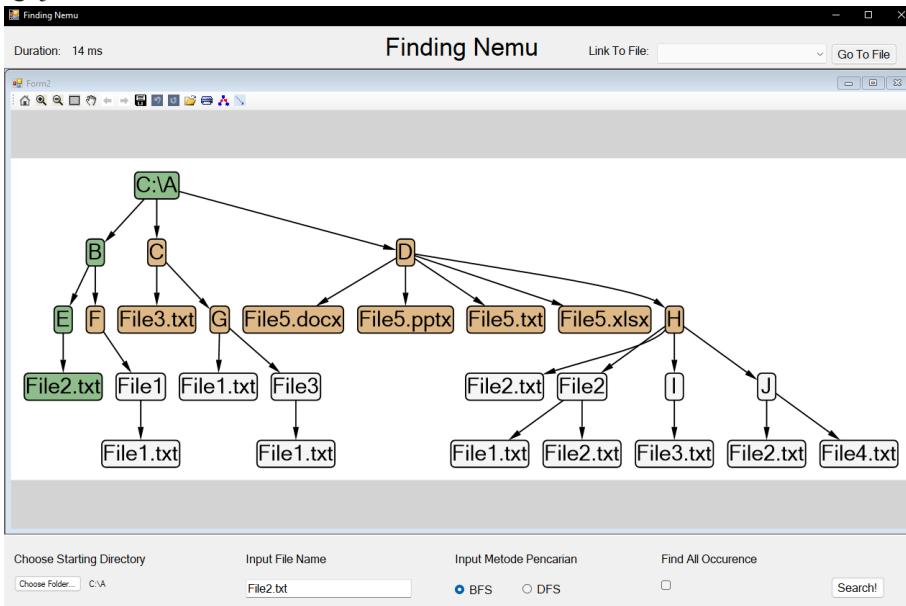
Tabel 4 Data uji 3

| <b>TESTING 3</b>              |   |
|-------------------------------|---|
| Skenario Pengujian            | <ul style="list-style-type: none"> <li>• Menggunakan DFS</li> <li>• Tidak All Occurrence</li> <li>• File tidak ditemukan</li> </ul> |
| Root Folder                   | C:\A  |
| Nama file yang dicari         | hambin.txt  |
| Hasil Pengujian:              |   |
| <p>Waktu pencarian: 13 ms</p> |   |

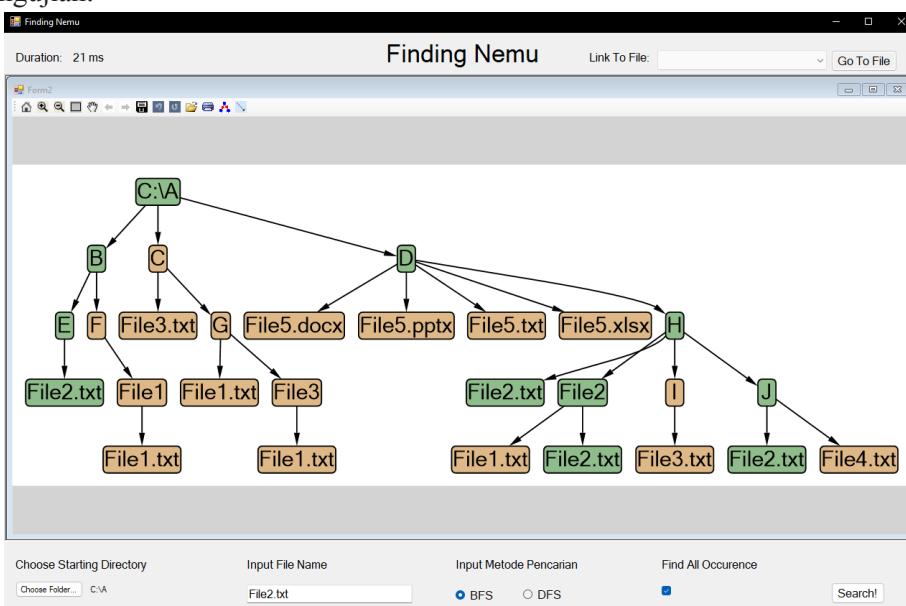
Tabel 5 Data uji 4

| <b>TESTING 4</b>              |   |
|-------------------------------|---|
| Skenario Pengujian            | <ul style="list-style-type: none"> <li>• Menggunakan DFS</li> <li>• All Occurrence</li> <li>• File tidak ditemukan</li> </ul> |
| Root Folder                   | C:\A  |
| Nama file yang dicari         | hambin.txt  |
| Hasil Pengujian:              |   |
| <p>Waktu pencarian: 14 ms</p> |   |

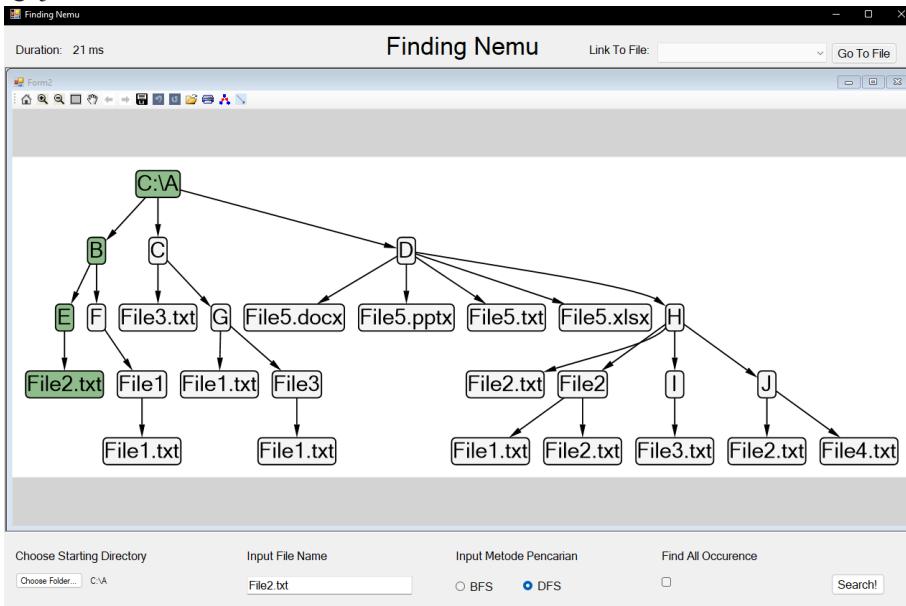
Tabel 6 Data uji 5

| <b>TESTING 5</b>  |   |
|---|---|
| Skenario Pengujian  | <ul style="list-style-type: none"> <li>• Menggunakan BFS</li> <li>• Tidak All Occurrence</li> <li>• File ditemukan</li> </ul> |
| Root Folder   | C:\A  |
| Nama file yang dicari   | File2.txt   |
| Hasil Pengujian:  |   |
|  <p>Waktu pencarian: 14 ms</p> |   |

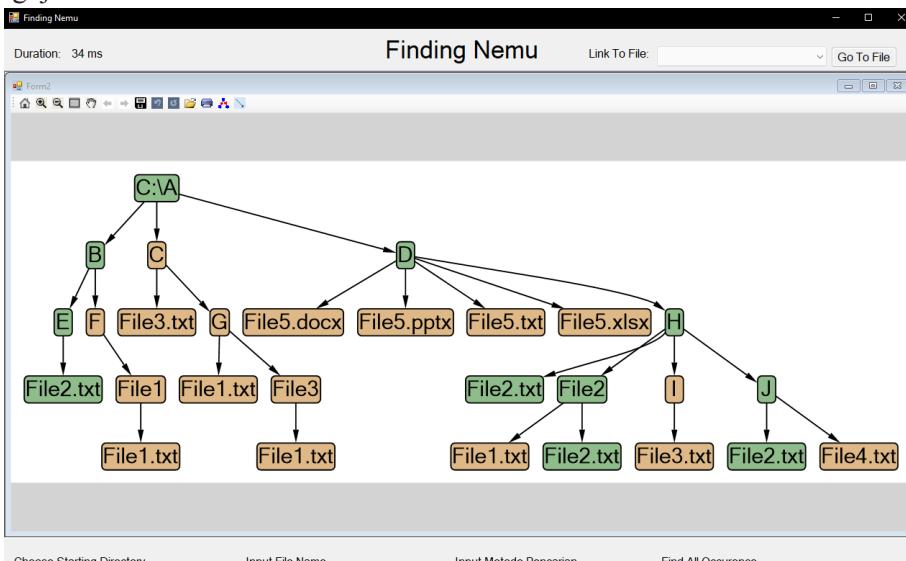
Tabel 7 Data uji 6

| <b>TESTING 6</b>   |   |
|--|---|
| Skenario Pengujian   | <ul style="list-style-type: none"> <li>• Menggunakan BFS</li> <li>• All Occurrence</li> <li>• File ditemukan</li> </ul> |
| Root Folder  | C:\A  |
| Nama file yang dicari  | File2.txt   |
| Hasil Pengujian:   |   |
|  <p>Waktu pencarian: 14 ms</p> |   |

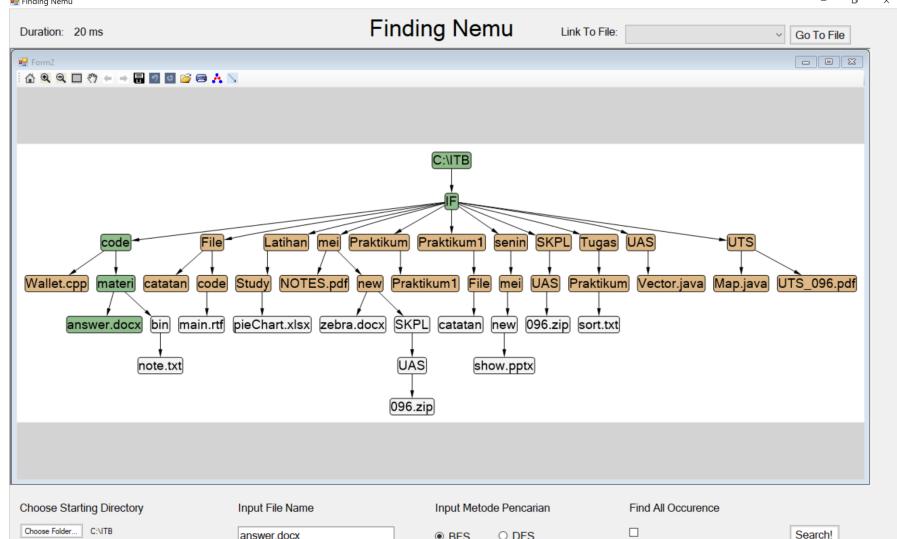
Tabel 8 Data uji 7

| <b>TESTING 7</b>  |   |
|---|---|
| Skenario Pengujian  | <ul style="list-style-type: none"> <li>• Menggunakan DFS</li> <li>• Tidak All Occurrence</li> <li>• File ditemukan</li> </ul> |
| Root Folder   | C:\A  |
| Nama file yang dicari   | File4.txt   |
| Hasil Pengujian:  |   |
|  <p>Waktu pencarian: 21 ms</p> |   |

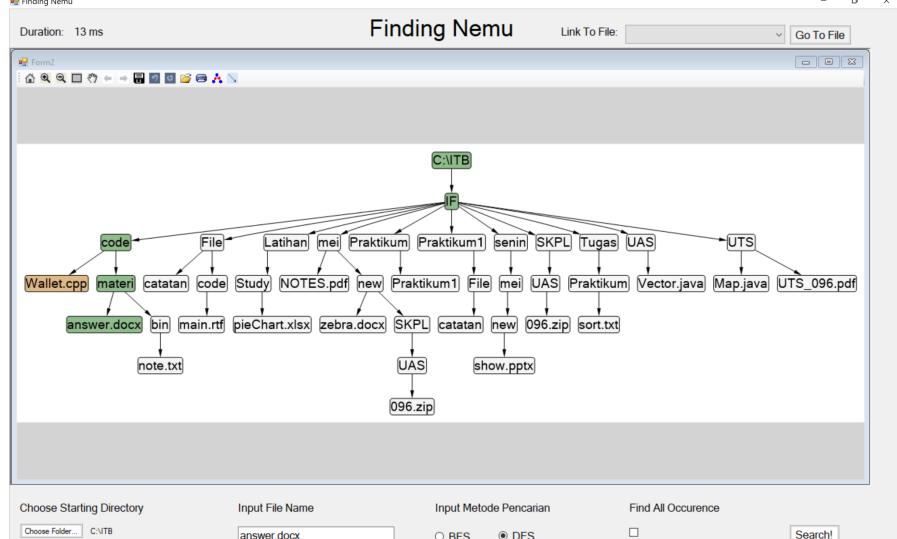
Tabel 9 Data uji 8

| <b>TESTING 8</b>   |   |
|--|---|
| Skenario Pengujian   | <ul style="list-style-type: none"> <li>• Menggunakan DFS</li> <li>• All Occurrence</li> <li>• File ditemukan</li> </ul> |
| Root Folder  | C:\A  |
| Nama file yang dicari  | File2.txt   |
| Hasil Pengujian:   |   |
|  <p>Waktu pencarian: 34 ms</p> |   |

Tabel 10 Data uji 9

| <b>TESTING 9</b>  |   |
|---|---|
| Skenario Pengujian  | <ul style="list-style-type: none"> <li>• Menggunakan BFS</li> <li>• Tidak All Occurrence</li> <li>• File ditemukan</li> <li>• Folder melebar tapi tidak mendalam</li> </ul> |
| Root Folder   | C:\ITB  |
| Nama file yang dicari   | answer.docx   |
| Hasil Pengujian:  |   |
|  <p>Waktu pencarian: 20 ms</p> |   |

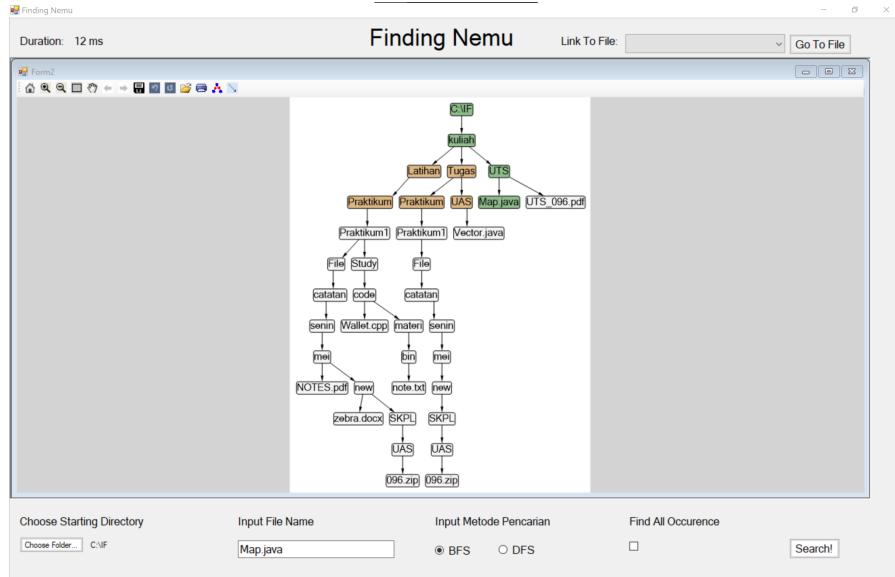
Tabel 11 Data uji 10

| <b>TESTING 10</b>  |   |
|--|---|
| Skenario Pengujian   | <ul style="list-style-type: none"> <li>• Menggunakan DFS</li> <li>• Tidak All Occurrence</li> <li>• File ditemukan</li> <li>• Folder melebar tapi tidak mendalam</li> </ul> |
| Root Folder  | C:\ITB  |
| Nama file yang dicari  | answer.docx   |
| Hasil Pengujian:   |   |
|  <p>Waktu pencarian: 13 ms</p> |   |

Tabel 12 Data uji 11

| TESTING 11            |   |
|-----------------------|---|
| Skenario Pengujian    | <ul style="list-style-type: none"> <li>• Menggunakan BFS</li> <li>• Tidak All Occurrence</li> <li>• File ditemukan</li> <li>• Folder tidak melebar tapi mendalam</li> </ul> |
| Root Folder           | C:\IF   |
| Nama file yang dicari | Map.java  |

Hasil Pengujian:

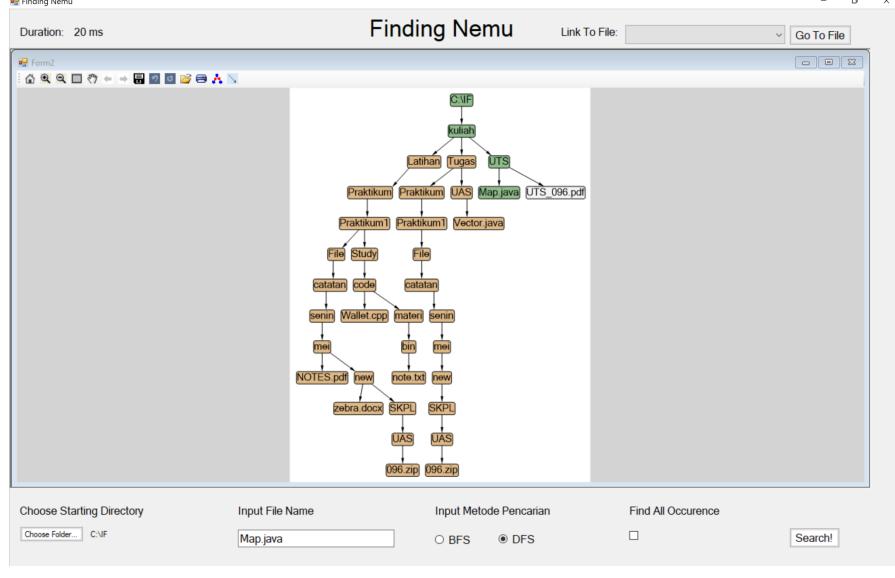


Waktu pencarian: 12 ms

Tabel 13 Data uji 12

| TESTING 12            |   |
|-----------------------|---|
| Skenario Pengujian    | <ul style="list-style-type: none"> <li>• Menggunakan DFS</li> <li>• Tidak All Occurrence</li> <li>• File ditemukan</li> <li>• Folder tidak melebar tapi mendalam</li> </ul> |
| Root Folder           | C:\IF   |
| Nama file yang dicari | Map.java  |

Hasil Pengujian:



Waktu pencarian: 20 ms

## G. Analisis Desain Solusi

Perbedaan paling dasar dari algoritma BFS dan DFS terlihat dari namanya, dimana BFS akan meng-iterasi semua simpul yang bertetangga dengan simpul awal, sedangkan DFS akan mengunjungi semua simpul yang terhubung ke simpul awal dan apabila sudah mencapai ujung, melakukan backtrack kembali ke simpul awal. Penentuan algoritma yang lebih baik untuk digunakan dapat dilihat dalam penjabaran keuntungan berikut ini.

Keuntungan BFS dari DFS adalah sifatnya yang meng-iterasi semua simpul yang terhubung (bertetangga) dengan simpul awal, yang artinya BFS akan selalu menemukan solusi. Apabila struktur folder yang digunakan cukup kompleks (memiliki banyak simpul), maka BFS akan selalu menemukan solusi, walaupun waktu yang dibutuhkan lebih lama karena BFS akan mengecek setiap tetangga sebuah simpul.

Keuntungan DFS dari BFS adalah sifatnya yang mengunjungi simpul yang terhubung dengan simpul awal sampai ujung. Artinya DFS akan cepat menemukan solusi apabila letak solusi berada di simpul kiri bawah. DFS cocok untuk digunakan pada struktur folder yang tidak melebar tetapi mendalam (tidak memiliki banyak anak folder) karena DFS akan mencari hingga ujung simpul. Akan tetapi, DFS tidak akan selalu menemukan solusi, misalnya untuk struktur folder yang melebar dan sangat mendalam, karena DFS akan terus mencari hingga ujung simpul.

Dari hasil pengujian yang sudah dilakukan, perbedaan antara strategi BFS dan DFS sangat terlihat pada Testing 9, 10, 11, dan 12. Pada Testing 9 dan 10, *starting/root folder* memiliki struktur *graph / pohon* yang melebar tetapi tidak mendalam. Sedangkan Testing 11 dan 12, *starting/root folder* memiliki struktur *graph/pohon* yang tidak melebar namun mendalam. Folder yang digunakan pada testing 9 dan 10 merupakan folder yang sama dan file yang dicari pun sama. Namun, terdapat perbedaan waktu pencarian file. Dengan metode BFS membutuhkan waktu 20 ms sedangkan DFS membutuhkan waktu 13 ms. Seperti penjelasan sebelumnya, solusi terletak pada simpul kiri bawah, sehingga algoritma DFS lebih diuntungkan dibandingkan BFS. Begitu juga pada testing 11 dan 12. Folder yang digunakan pada testing 1 dan 12 merupakan folder yang sama dan file yang dicari pun sama. Namun, terdapat perbedaan waktu pencarian file. Dengan metode BFS membutuhkan waktu 12 ms sedangkan DFS membutuhkan waktu 20 ms. Seperti penjelasan sebelumnya, solusi terletak pada simpul yang berdekatan dengan *root* sehingga dapat menemukan file lebih cepat dibandingkan DFS yang mencari sampai ujung simpul terlebih dahulu.

## **BAB V**

### **Kesimpulan**

#### **A. Kesimpulan**

Berdasarkan hasil pembuatan *Folder Crawling* Finding Nemu dan analisisnya, dapat diambil kesimpulan bahwa struktur data *graph* terutama jenis *tree/pohon* dapat merepresentasikan folder beserta filenya dengan baik. Pemanfaatan struktur *tree* ini juga memudahkan proses traversal dari tiap-tiap simpul yang ada. Proses traversal ini digunakan dalam pencarian file dengan menggunakan algoritma BFS maupun DFS.

Metode pencarian dengan menggunakan BFS dan DFS dapat diterapkan pada proses *folder crawling* pencarian file pada suatu folder. Kedua algoritma dapat menemukan solusi dengan tepat dan membutuhkan waktu yang kurang lebih sama. Perbedaan dari kedua algoritma terlihat pada penelusuran node, sehingga jalur yang dilalui saat proses traversal berbeda. Urutan file yang dikunjungi menjadi berbeda. BFS dan DFS masing-masing unggul di kondisi tertentu. BFS untuk ketika *graph* melebar tapi tidak mendalam. Sedangkan DFS unggul untuk kondisi *graph* yang tidak melebar, namun mendalam.

#### **B. Saran**

Saran yang dapat kami berikan bagi pembaca dari penggerjaan Tugas Besar IF2211 Strategi Algoritma semester 2 2021/2022 adalah:

1. Melakukan eksplorasi lebih dalam lagi terkait C# Desktop Application Development. Mulai dari struktur file yang ada pada aplikasi ini, cara kerja, maupun kreatifitas dalam pembuatan *graphical user interface* (GUI).
2. Melengkapi *code program* dengan komentar, agar dalam pengembangan atau *debug* dapat memahami *code* dengan mudah.

## Daftar Pustaka

- Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2021). “Breadth/Depth First Search (BFS/DFS) Bagian 1”. Diakses online dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf> pada 16 Maret 2022.
- Munir, Rinaldi. (2020). “Pohon Bagian 1”. Diakses online dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf> pada 16 Maret 2022.
- Unknown. (2019). “Mengenal Algoritma Traversal di dalam Graf”. Diakses online dari <https://www.ilmuskripsi.com/2016/05/mengenal-algoritma-traversal-di-dalam.html> pada 16 Maret 2022.
- Shiddiq, Mohammad Mahfuzh. (2018). “Graf Pohon - Teori Graf”. Diakses online dari <https://www.haimatematika.com/2018/12/graf-pohon-teori-graf.html> pada 16 Maret 2022.