LAPORAN TUGAS BESAR 2

IF2211 Strategi Algoritma

Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder Crawling*

Disusun oleh:

Kelompok 8 Digging Deep

13520024 – Hilya Fadhilah Imania

13520051 – Flavia Beatrix Leoni A. S.

13520129 – Nathanael Santoso

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

Daftar Isi

Daftar Gambar

BAB 1

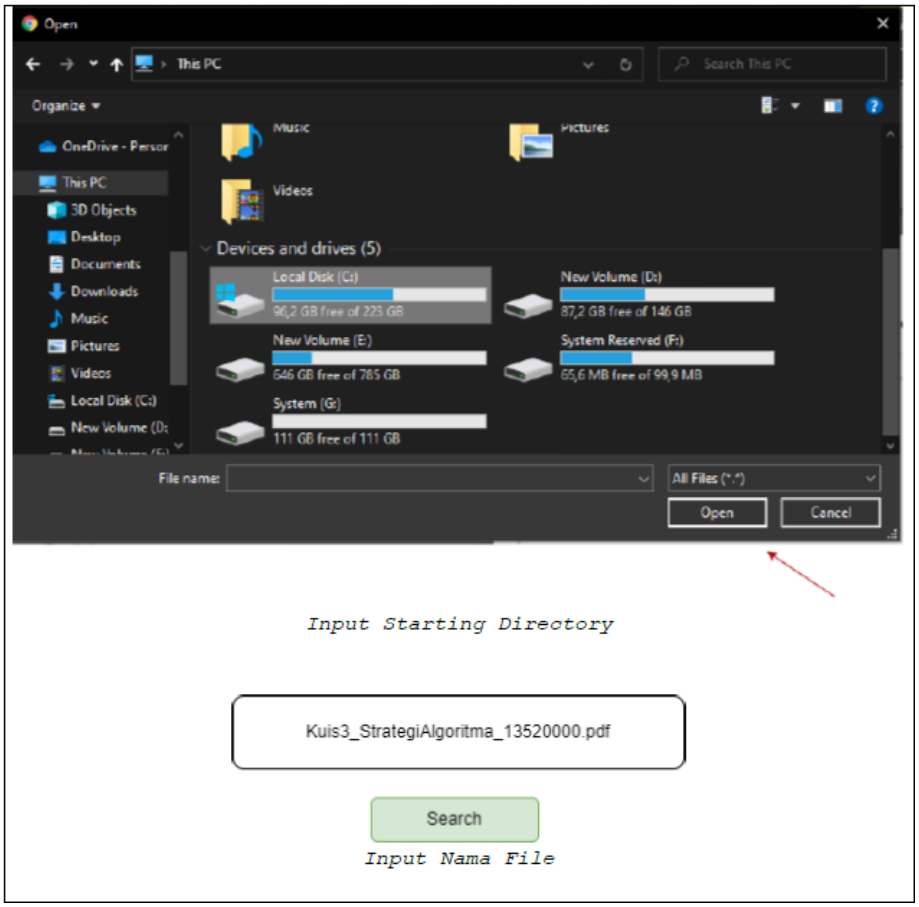
Deskripsi Masalah

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 harus 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.

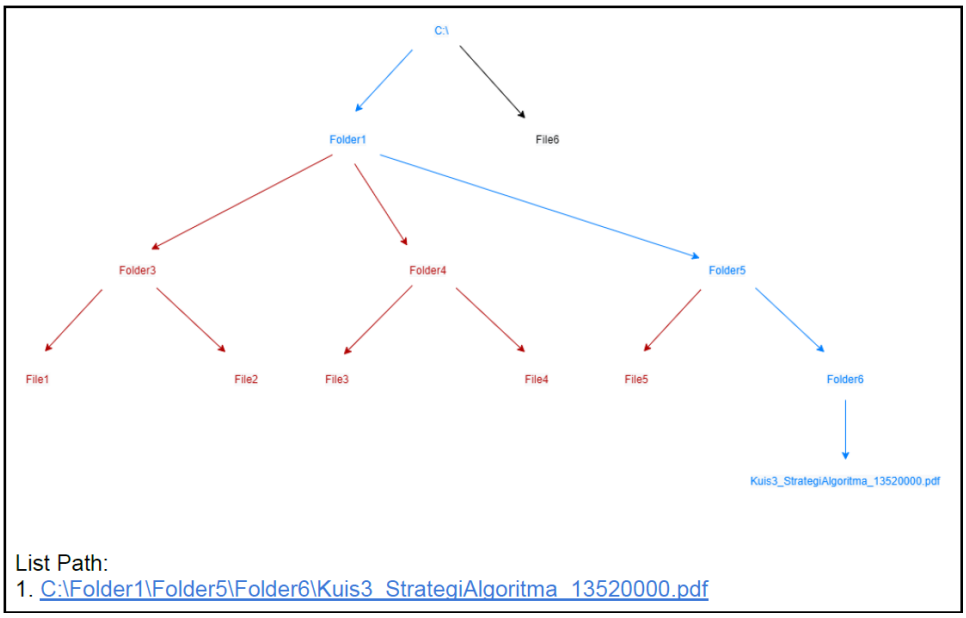
Contoh Input dan Output Program

Contoh masukan aplikasi:



Gambar 1.1 Contoh input program

Contoh output aplikasi:

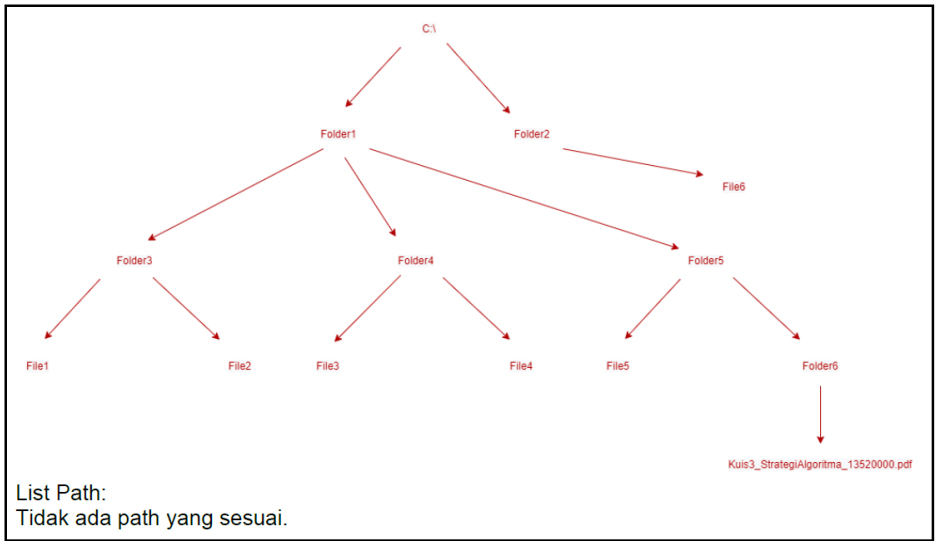


Gambar 1.2 Contoh output program

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 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.

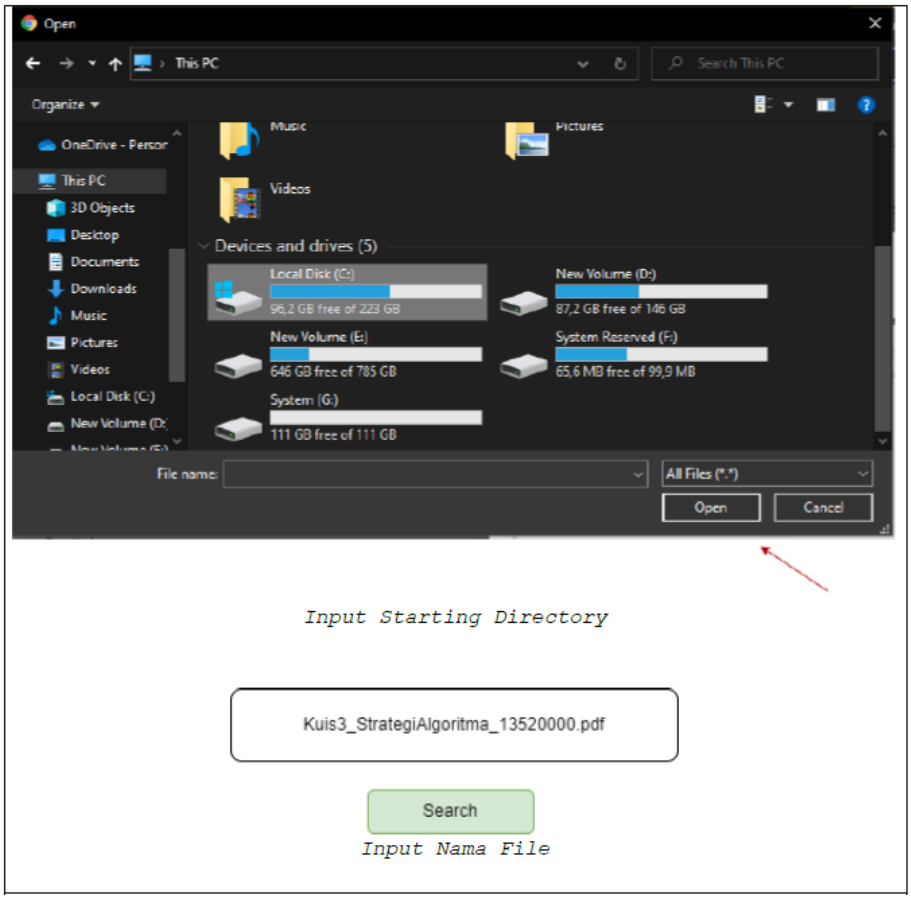


Gambar 1.3 Contoh output program jika file tidak ditemukan

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 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 1.4 Contoh ketika hyperlink di-klik

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

Selain pohon, Anda diminta juga menampilkan list path dari daun-daun yang bersesuaian dengan hasil pencarian. Path tersebut diharuskan memiliki hyperlink menuju folder parent dari file yang dicari, agar file langsung dapat diakses melalui browser atau file explorer.

BAB 2
Landasan Teori

2.1 Graf traversal, BFS, dan DFS

Graf merupakan representasi yang digunakan dalam pencarian solusi suatu persoalan. Pencarian solusi persoalan dengan algoritma traversal graf dilakukan dengan cara mengunjungi simpul dengan cara yang sistematik. Jenis-jenis algoritma graf traversal antara lain *Breadth First Search* (BFS) dan *Depth First Search* (DFS).

2.1.1 *Breadth First Search* (BFS)

BFS merupakan metode pencarian yang bersifat melebar, yaitu memprioritaskan melihat ke tetangga dari suatu cabang yang sedang diperiksa. Langkahnya sebagai berikut:

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.

2.1.2 *Depth First Search* (DFS)

Depth First Search adalah suatu metode pencarian pada sebuah pohon dengan menelusuri satu cabang sebuah pohon sampai menemukan solusi. Algoritma DFS yang dilakukan secara traversal dimulai dari simpul *v* adalah sebagai berikut.

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 dirunut-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 dicapai dari simpul yang telah dikunjungi.

2.2 C# Desktop Application Development

2.2.1 Pendahuluan

Pada tugas besar ini, pengembangan aplikasi berbasis *Graphical User Interface* (GUI) dilakukan dengan bahasa C# (pengejaan "see sharp") menggunakan *software framework* .NET (pengejaan "dot net"). *Software framework* merupakan abstraksi dari pemrograman level rendah menjadi berbagai macam fungsi yang dapat dipanggil, digunakan, dan dimodifikasi oleh seorang pengembang. Berbeda dengan *library*, *software framework* mengubah alur pemrograman sehingga pengembang tidak lagi menentukan keberjalanan program melainkan *software framework* yang menentukan. Hal ini menguntungkan karena pengembang bisa fokus untuk memenuhi kebutuhan perangkat lunak dengan lebih mudah dan cepat.

2.2.2 Software *Framework* .NET

Software framework .NET merupakan *software framework* gratis dan *open-source* yang dikembangkan oleh Microsoft untuk pengguna Windows, namun memiliki fungsi lintas platform sehingga dapat dijalankan juga di Linux dan MacOS. Versi terbaru dari .NET adalah .NET 6 yang digunakan pada aplikasi ini. Versi ini mendukung bahasa C#, F#, C++, dan Visual Basic dan menyediakan pemrograman lintas bahasa menggunakan Common Language Infrastructure (CLI) sehingga program dapat ditulis dalam berbagai bahasa dan tetap dapat dijalankan. Pada .NET, CLI diimplementasikan sebagai CoreCLR yang menyediakan kegunaan mirip dengan *Common Language Runtime* (CLR) milik .NET Framework (pendahulu .NET) dan CoreFX yang menyediakan berbagai *standard library* mirip dengan pendahulunya *Framework Class Library* (FCL) milik .NET Framework. CoreFX menyediakan beberapa *library* untuk pembangunan GUI seperti Windows Presentation Foundation dan Windows Forms yang digunakan pada aplikasi ini. Selain itu, .NET juga menyediakan alat untuk memakai *library* luar melalui *package manager* bernama NuGet. *Library* luar yang digunakan dalam aplikasi ini adalah *Microsoft Automatic Graph Layout* (MSAGL) dan FontAwesome.sharp.

2.2.3 Bahasa C#

Bahasa C# adalah bahasa pemrograman sederhana yang digunakan untuk tujuan umum, dalam artian bahasa pemrograman ini dapat digunakan untuk berbagai fungsi misalnya untuk pemrograman server-side pada website, membangun aplikasi desktop atau mobile, pemrograman game, dan sebagainya. Selain itu, C# juga bahasa pemrograman yang berorientasi objek, jadi C# juga mengusung konsep objek seperti *inheritance*, *class*, *polymorphism*, dan *encapsulation*.

Terdapat 5 struktur dasar pada pemrograman C#, yaitu:

1. *Resource* atau *library*, merupakan pendefinisian *library* apa yang harus ada pada program
2. Namespace, merupakan nama dari project yang dibuat
3. Nama class
4. Deklarasi Method, merupakan pendeklarasian method sebagai awalan untuk menjalankan method atau perintah yang ada di dalamnya
5. *Method* atau *Command*, merupakan *method* atau perintah yang diberikan untuk dieksekusi oleh compiler

BAB 3
Analisis Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

Secara garis besar, langkah-langkah pemecahan masalah yang dilakukan adalah sebagai berikut.

1. Menerima masukan berupa folder awal pencarian dan menyimpannya sebagai simpul-simpul dari suatu pohon. Pada tiap simpul, disimpan nama folder/file, id, dan list anak dari simpul tersebut.
2. Menerima masukan berupa nama file yang ingin dicari, pilihan jenis algoritma yang digunakan (BFS/DFS), dan pilihan pencarian, yaitu mencari 1 file saja atau mencari semua kemunculan file pada folder root.
3. Apabila pilihan pengguna adalah BFS, pencarian akan dilakukan dengan mengunjungi semua simpul yang bertetangga dengan simpul tersebut terlebih dahulu, kemudian dilanjutkan dengan simpul yang belum dikunjungi dan bertetangga dengan simpul yang telah dikunjungi. Sementara, apabila pilihan pengguna adalah DFS, pencarian akan dilakukan dengan mengunjungi simpul hingga mencapai level terdalam terlebih dahulu, kemudian *backtrack* ke simpul sebelumnya. Apabila pengguna memilih untuk mencari 1 file saja, maka pencarian akan berakhir setelah file tersebut ditemukan. Namun, pencarian akan dilakukan hingga mengunjungi semua simpul yang ada jika pengguna memilih untuk mencari semua kemunculan file.
4. Setelah proses pencarian selesai, akan ditampilkan visualisasi pohon pencarian file berdasarkan informasi masukan direktori beserta keterangan tiap simpul. Simpul yang sudah masuk antrian tetapi belum diperiksa akan berwarna hitam, simpul yang sudah diperiksa akan berwarna merah, dan simpul yang merupakan bagian dari rute hasil pencarian akan berwarna biru. Selain itu, akan ditampilkan juga durasi waktu algoritma dan *hyperlink* pada setiap hasil rute yang ditemukan yang akan membuka lokasi file pada *file explorer*.

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

Secara umum pada penyelesaian, filesystem direpresentasikan dengan sebuah pohon. Simpul-simpul menyatakan nama direktori atau file. Setiap upapohon merupakan sebuah direktori dengan simpul anak berupa direktori atau file yang berada dalam direktori tersebut.

Akar atau *root* dari pohon adalah direktori pencarian yang diinput oleh pengguna. Fungsi solusi dari persoalan merupakan path atau jalur dari akar ke simpul yang memenuhi kriteria, yaitu memiliki nama yang sama dengan nama file yang dicari.

Pada algoritma BFS, proses pencarian dilakukan dengan menggunakan antrian atau queue, yaitu struktur data dengan prinsip *First-In-First-Out* (FIFO). Karena itu, algoritma dapat diimplementasikan secara iteratif. Untuk setiap simpul yang sedang ditinjau, jika ia berupa upapohon maka simpul anaknya akan dimasukkan ke dalam antrian periksa. Jika simpul yang ditinjau bernilai sama dengan kriteria pencarian, maka simpul tersebut merupakan solusi.

Pada algoritma DFS, proses pencarian dilakukan secara rekursif dan dimulai dari simpul direktori masukan. Untuk setiap langkah rekursif, simpul yang telah dikunjungi akan ditandai dan urutan simpul yang dilewati akan dicatat. Pada setiap simpul, akan dicek apakah simpul tersebut merupakan simpul yang dicari. Jika ya dan pengguna memilih untuk mencari 1 file saja, maka proses pencarian akan dihentikan dan rute menuju simpul tersebut dicatat, tetapi jika pengguna memilih untuk mencari semua kemunculan file, maka proses pencarian akan dilanjutkan ke simpul yang bertetangga dengan simpul tersebut. Langkah ini dilanjutkan hingga pencarian telah mencapai simpul daun. Apabila belum ditemukan, pencarian akan mundur ke simpul sebelumnya dan dilanjutkan dengan memeriksa simpul tetangga yang belum dikunjungi. Proses ini diulang hingga simpul yang dicari telah ditemukan atau semua simpul telah dikunjungi.

3.3 Ilustrasi Kasus

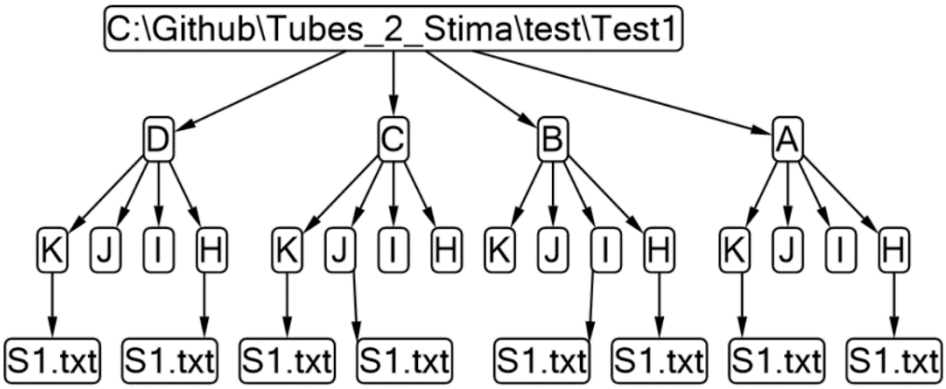
3.3.1 Ilustrasi Kasus Pertama

Untuk input direktori:

```
|-- C:\Github\Tubes_2_Stima\test\Test1
  |-- A
    |-- H
      | S1.txt
    |-- I
    |-- J
    |-- K
```

```
    | S1.txt
|-- B
  |-- H
    | S1.txt
  |-- I
    | S1.txt
  |-- J
  |-- K
|-- C
  |-- H
  |-- I
  |-- J
    | S1.txt
  |-- K
    | S1.txt
|-- D
  |-- H
    | S1.txt
  |-- I
  |-- J
  |-- K
    | S1.txt
```

Akan Mengeluarkan Pohon:



Gambar 3.1 Pohon Direktori Kasus Pertama

3.3.2 Ilustrasi Kasus Kedua

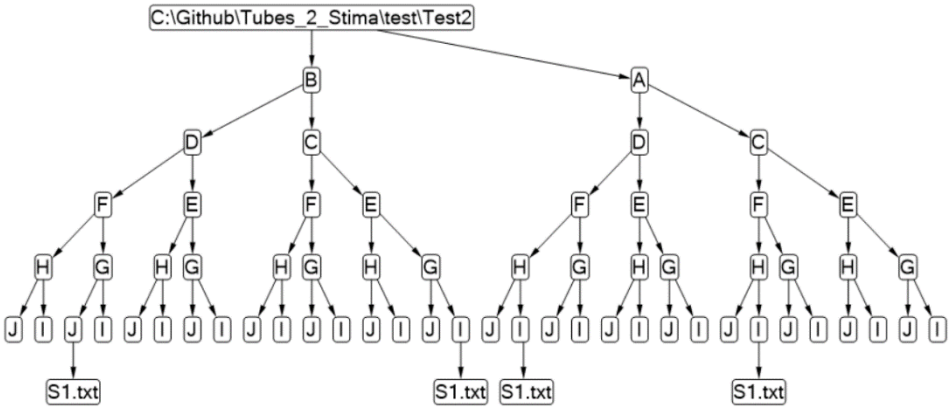
Untuk Input Direktori:

```
|-- C:\Github\Tubes_2_Stima\test\Test2
  |-- A
    |-- C
      |-- E
        |-- G
          |-- I
          |-- J
        |-- H
          |-- I
          |-- J
      |-- F
        |-- G
          |-- I
          |-- J
        |-- H
          |-- I
            | S1.txt
```

```
        |-- J
    |-- D
      |-- E
        |-- G
          |-- I
          |-- J
        |-- H
          |-- I
          |-- J
      |-- F
        |-- G
          |-- I
          |-- J
        |-- H
          |-- I
            | S1.txt
          |-- J
|-- B
  |-- C
    |-- E
      |-- G
        |-- I
          | S1.txt
        |-- J
      |-- H
```

```
          |-- I
          |-- J
      |-- F
        |-- G
          |-- I
          |-- J
        |-- H
          |-- I
          |-- J
    |-- D
      |-- E
        |-- G
          |-- I
          |-- J
        |-- H
          |-- I
          |-- J
      |-- F
        |-- G
          |-- I
          |-- J
            | S1.txt
        |-- H
          |-- I
          |-- J
```

Akan Mengeluarkan Pohon:



Gambar 3.2 Pohon Direktori Kasus Kedua

BAB 4

Implementasi dan Pengujian

4.1 Implementasi Program

Berikut merupakan pseudocode dari algoritma-algoritma yang penting dalam pembuatan aplikasi ini.

a. Program Utama

```
procedure main()
    KAMUS LOKAL
    stopwatch : Stopwatch
    foldername, filename, path : string
    graph : TreeNode
    res : HashSet of string

    ALGORITMA
    input(foldername)
    IterateDirectory(foldername, graph)
    input(filename)

    stopwatch.Start()
    if (BFSChecked) then
        res ← bfs.Search(filename, checkboxChecked)
    else if (DFSChecked) then
        res ← dfs.Search_DFS(filename, checkboxChecked)

    if (res.Count > 0) then
        foreach (path in res)
            linkPath.Links.Add(path)

    stopwatch.Stop()

procedure IterateDirectory(input string path, input TreeNode node)
    KAMUS LOKAL
    pathname, file, dir : string
    files, dirs : array of string
```

```
    child : TreeNode
    graphNode : Node

    ALGORITMA
    pathname ← GetFileName(path)
    files ← GetFiles(path)
    foreach (file in files)
        filename ← GetFileName(file)
        child ← node.AddChild(file, filename)
        graphNode ← graph.AddNode(file)
    dirs ← GetDirectories(path)
    foreach (dir in dirs)
        dirname ← GetFileName(dir)
        child ← node.AddChild(dir, dirname)
        graphNode ← graph.AddNode(dir)
        IterateDirectory(dir, child)
```

b. BFS

```
function Search(string filename, bool isFindAll) → HashSet of string
KAMUS LOKAL
    result : HashSet of string
    queue : Queue of BFSNode

ALGORITMA
    queue.Enqueue(BFSNode(tree))

    while (not queue.isEmpty) do
        current ← queue.Dequeue()
        if (current.Name = filename) then
            found.Add(current.Pathname)
            if (not isFindAll) then
                    queue.Clear()
        else
            foreach (child in current.Children) do
                queue.Enqueue(BFSNode(child, current.Path))

    → result
```

c. DFS

```
function Search_DFS (string filename, bool checkboxChecked) → result
KAMUS LOKAL
   result : HashSet of string
   path, visited : List of TreeNode
   found : boolean

ALGORITMA
   visited.Add(tree)
   path.Add(tree)
   Search_DFS_rec(filename, checkboxChecked, result, path, visited, found)
   → result

procedure Search_DFS_rec (input string filename, input bool checkboxChecked,
input/output HashSet result, List path, List visited, boolean found)
KAMUS LOKAL
   current : TreeNode
   pathFound : string

ALGORITMA
   if (path.Last().Name = filename) then
      found ← true
   else
      if (path.Last().Children.Count = 0) then
         path.Remove(path.Last())
      else
         current ← path.Last()
         foreach(child in current.Children)
            if (!visited.Contains(child)) then
               visited.Add(child)
               path.Add(child)
               Search_DFS_rec(filename, checkboxChecked, result, path, visited, found)
               if (found) then
                  pathFound ← path.First().Name
                  for i=1 to path.Count-1
                     pathFound ← pathFound + path[i].Name
```

```
                result.Add(pathFound)
                if (checkboxChecked) then
                    found ← false
                    path.Remove(path.Last())
                else
                    return
            else
                path.Remove(child)
```

4.2 Struktur Data yang Digunakan

Struktur data utama yang kami gunakan adalah TreeNode. TreeNode direpresentasikan dengan kelas yang memiliki atribut name, Id, dan Children. Atribut name merepresentasikan nama folder atau file yang disimpan pada node, sedangkan atribut Id merepresentasikan path dari folder atau file. Atribut Children merupakan sebuah List of TreeNode yang menyimpan semua node anak dari TreeNode tersebut. Untuk membuat sebuah objek dari kelas TreeNode, terdapat constructor dengan parameter id dan name. Kelas ini memiliki method AddChild yang berfungsi untuk membuat TreeNode baru untuk child dan memasukannya ke dalam List Children parent nya. Struktur data ini digunakan dalam proses pencarian dengan algoritma BFS dan DFS.

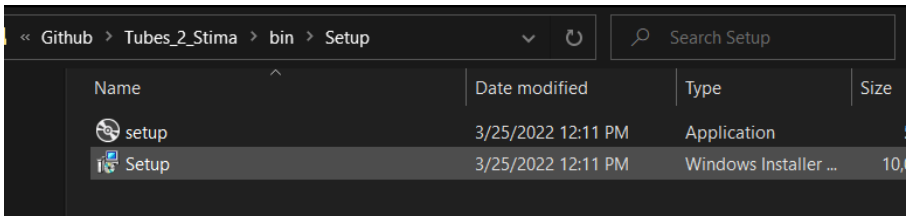
```
class TreeNode:
    {atribut}
    name : string
    Id : string
    Children : List of TreeNode


    {method}
    TreeNode(name, Id) {constructor}
    function AddChild(id: string, name: string) → TreeNode
    function ToString() → string
```

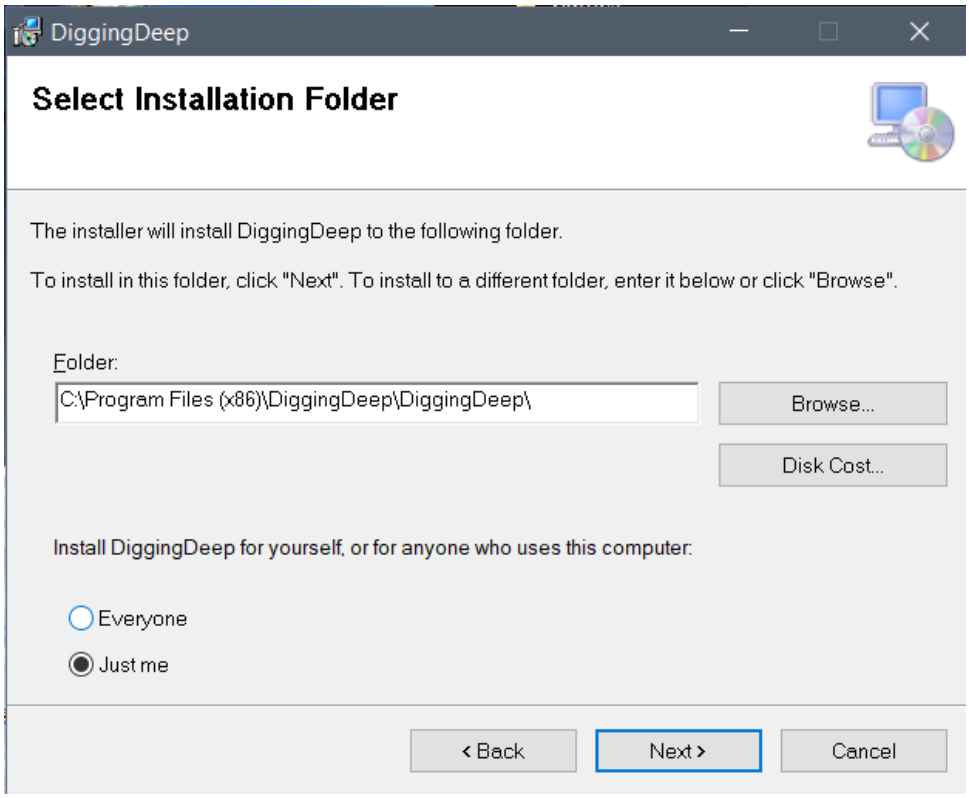
4.3 Tata Cara Penggunaan Program

4.3.1 Tata Cara Install

Jika ingin melakukan instalasi, dapat menggunakan Setup.exe, setelah memilih folder instalasi, akan menginstall DiggingDeep.exe dan membuat shortcut pada desktop bernama DiggingDeep, hanya perlu klik dua kali pada shortcut dan program akan dijalankan.



Gambar 4.1 Setup.exe

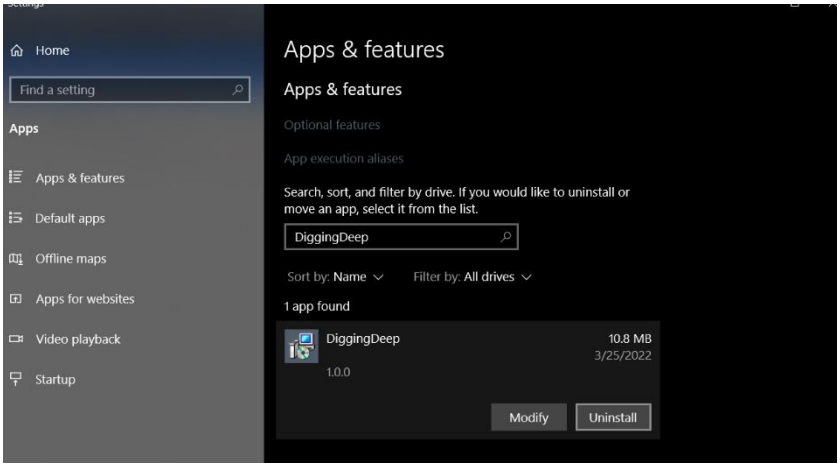


Gambar 4.2 Memilih Folder Install

Gambar 4.3 Shortcut Aplikasi yang Terletak di Desktop

4.3.2 Tata Cara Uninstall

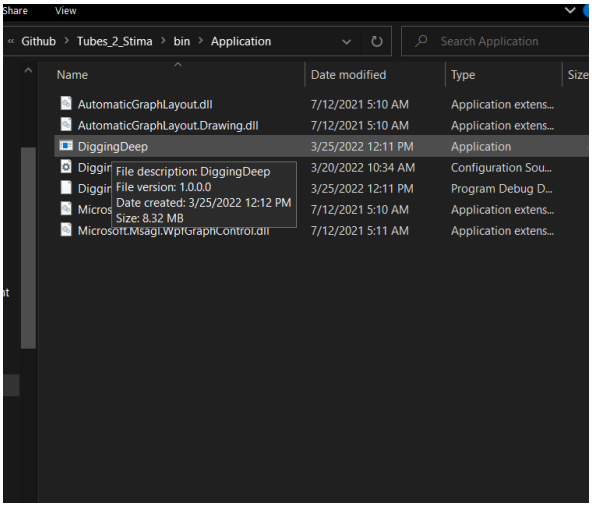
Untuk melakukan uninstall, cukup masuk ke app manager pada operating system yang dipakai, kemudian carilah aplikasi dengan nama DiggingDeep kemudian lakukan klik pada tombol uninstall.



Gambar 4.4 Meng-uninstall aplikasi DiggingDeep dari Sistem Operasi Windows 10

4.3.3 Tata Cara Penggunaan Aplikasi

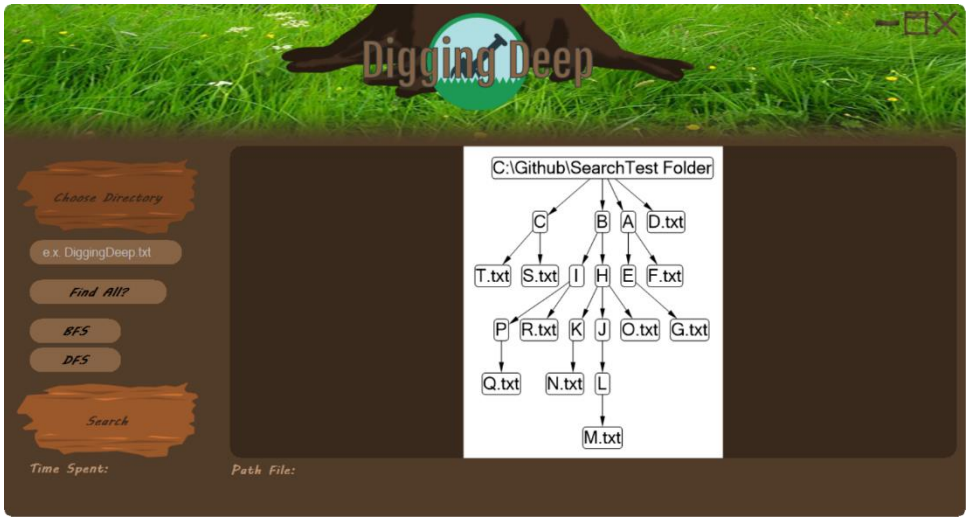
Untuk menggunakan aplikasi yang sudah diinstall, cukup menggunakan shortcut pada desktop yang sudah tersedia. Jika tidak ingin melakukan install, bisa juga membuka aplikasi dari DiggingDeep yang sudah tersedia.



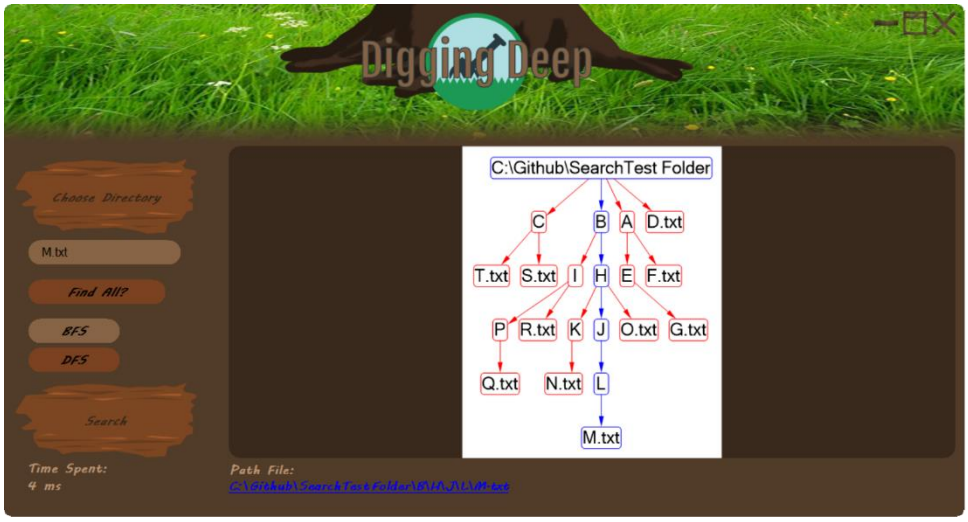
Gambar 4.5 Aplikasi DiggingDeep yang Tidak Perlu Diinstall

Gambar 4.6 Tampilan Aplikasi DiggingDeep

Jika sudah memasuki aplikasi, klik tombol “Choose Directory” untuk memilih folder yang akan dipakai untuk pencarian. Masukkan file yang ingin dicari pada textbox yang tersedia, kemudian pilih opsi pencarian. Setelah dipilih, klik tombol “Search” untuk memulai pencarian. Jika ada opsi yang belum terpilih saat menekan tombol “Search” akan dimunculkan pesan kesalahan. Setelah aplikasi menemukan file yang dituju, akan membuat link pergi ke direktori file. Jika link tersebut ditekan, akan membuka file yang dicari di *file explorer*. Karena keterbatasan UI, maka untuk opsi “Find All?” hanya akan menampilkan link dua pertama yang ditentukan.



Gambar 4.7 Tampilan Setelah Menekan Tombol “Choose Directory”

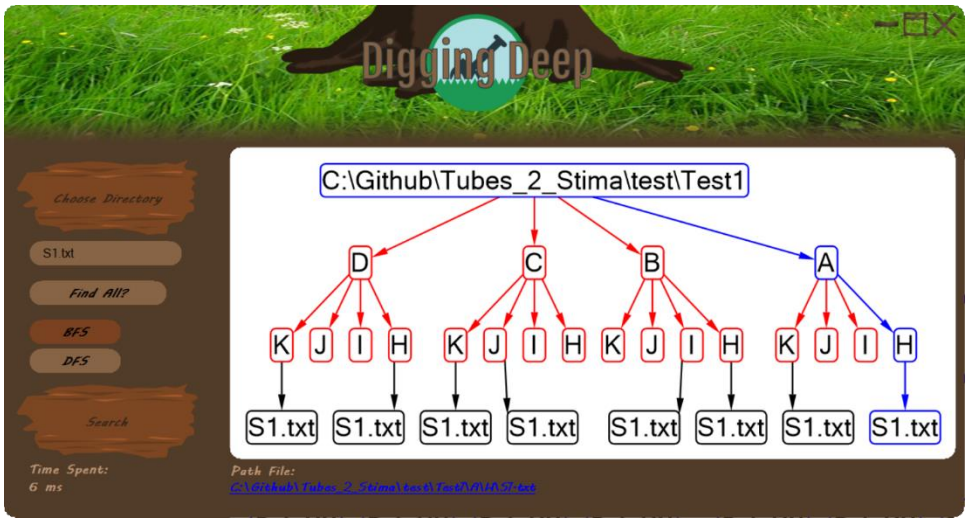


Gambar 4.8 Tampilan Setelah Menekan Tombol "Search"

4.4 Hasil Pengujian

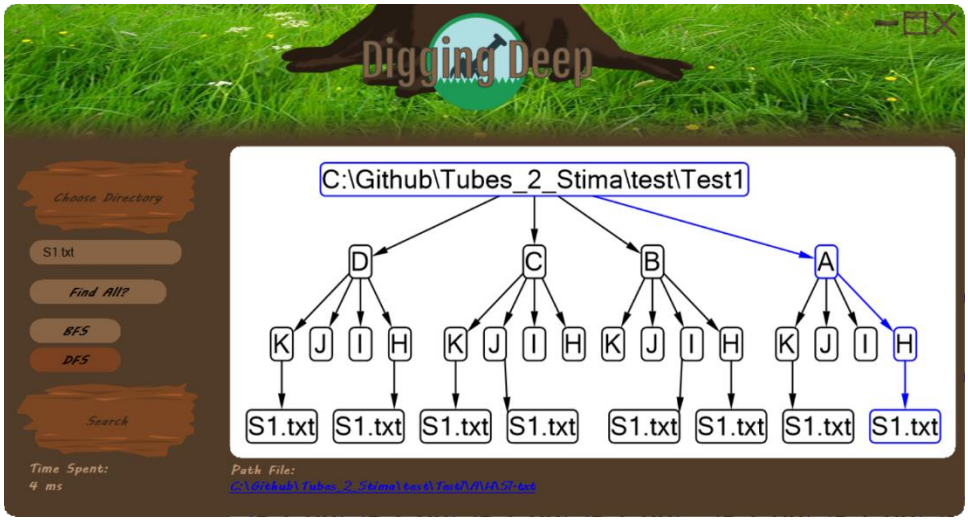
4.4.1 Hasil Pengujian Pertama

BFS:



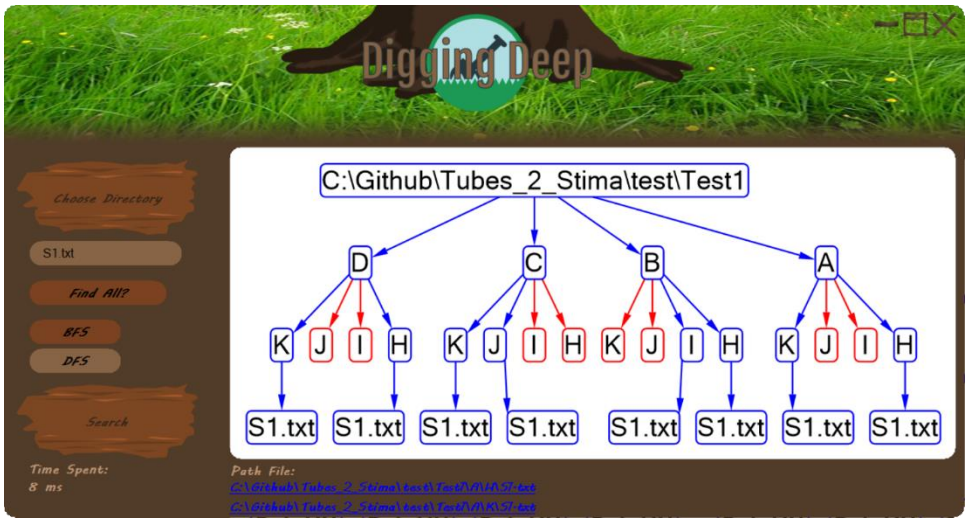
Gambar 4.9 Hasil Pengujian Pertama dengan Algoritma BFS

DFS:



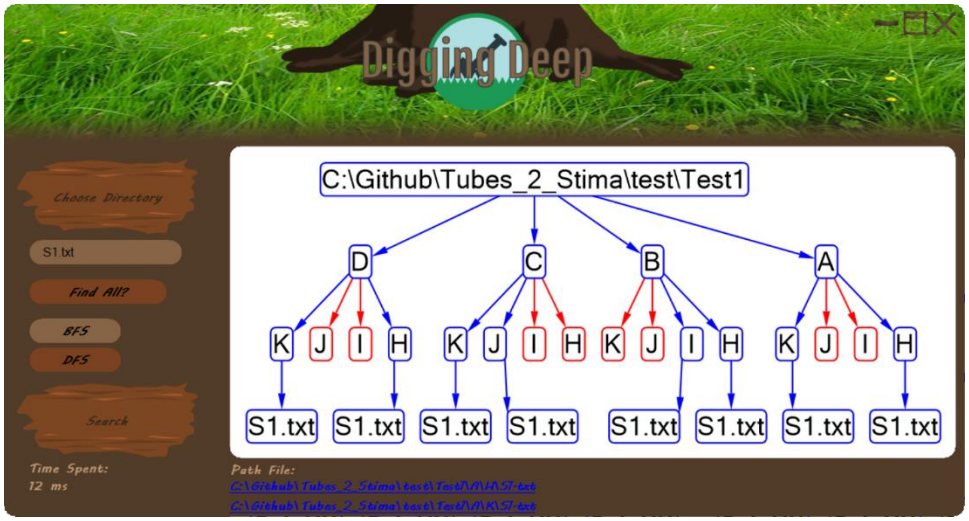
Gambar 4.10 Hasil Pengujian Pertama dengan Algoritma DFS

Find All BFS:



Gambar 4.11 Hasil Pengujian Pertama untuk Mencari Semua File dengan Algoritma BFS

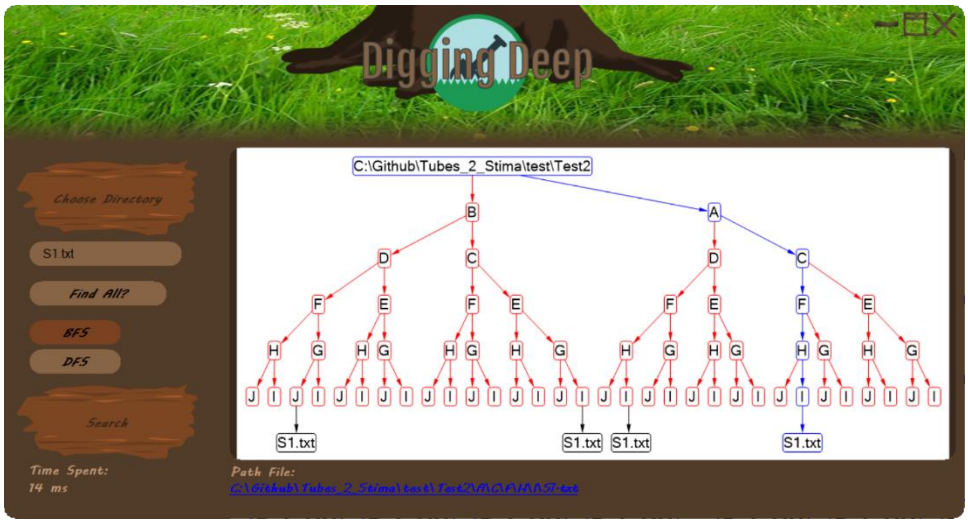
Find All DFS:



Gambar 4.12 Hasil Pengujian Pertama untuk Mencari Semua File dengan Algoritma DFS

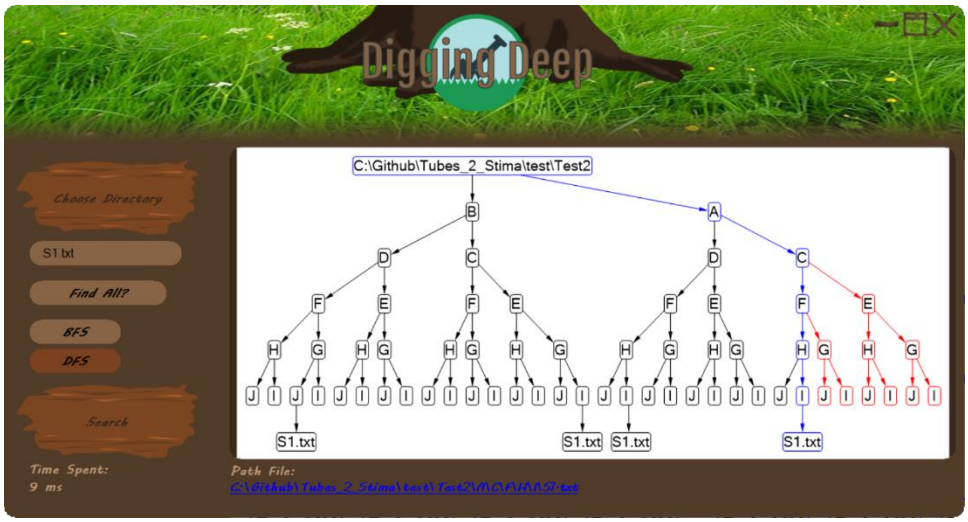
4.4.2 Hasil Pengujian Kedua

BFS:



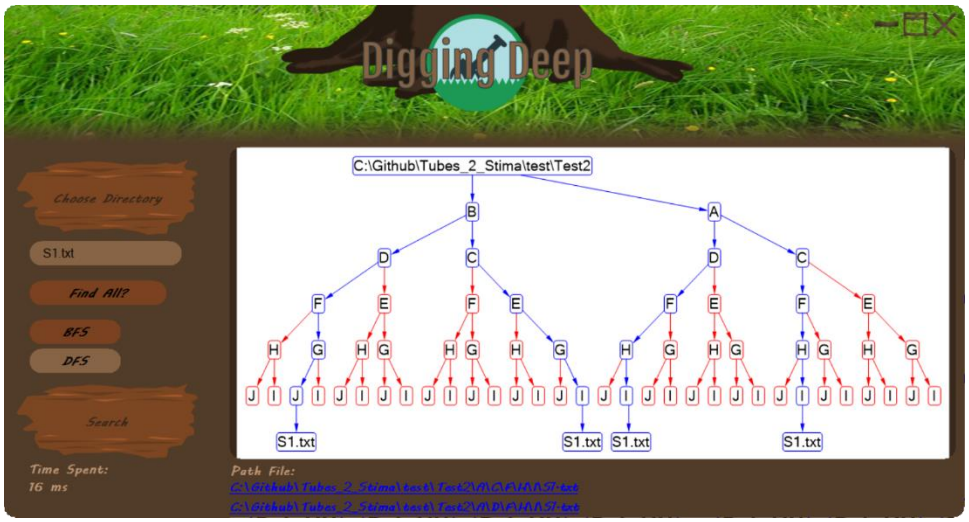
Gambar 4.13 Hasil Pengujian Kedua dengan Algoritma BFS

DFS:



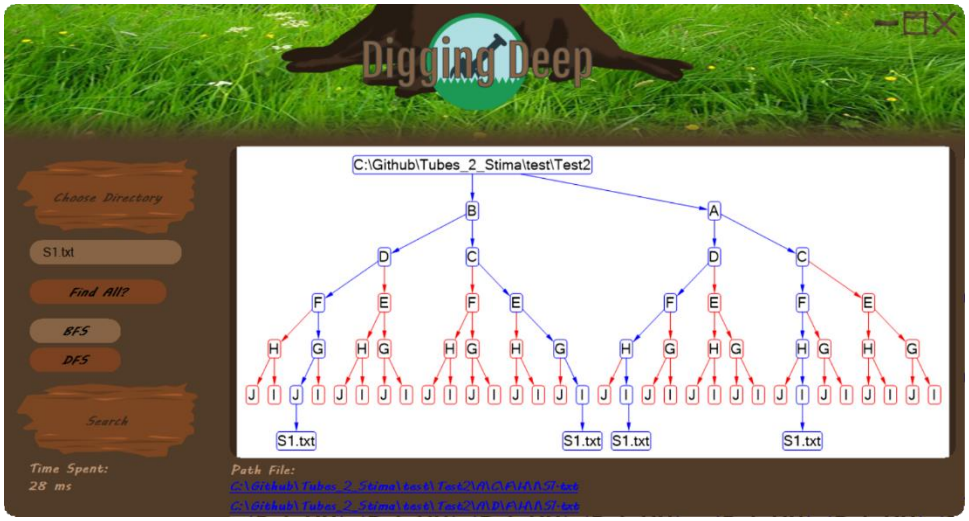
Gambar 4.14 Hasil Pengujian Kedua dengan Algoritma DFS

Find All BFS:



Gambar 4.15 Hasil Pengujian Kedua untuk Mencari Semua File dengan Algoritma BFS

Find All DFS:



Gambar 4.16 Hasil Pengujian Kedua untuk Mencari Semua File dengan Algoritma DFS

4.4.3 Hasil Pengujian Ketiga

BFS:

Gambar 4.17 Hasil Pengujian Ketiga dengan Algoritma BFS

DFS:

Gambar 4.18 Hasil Pengujian Ketiga dengan Algoritma DFS

Find All BFS:

Gambar 4.19 Hasil Pengujian Ketiga untuk Mencari Semua File dengan Algoritma BFS

Find All DFS:

Gambar 4.20 Hasil Pengujian Ketiga untuk Mencari Semua File dengan Algoritma DFS

4.5 Analisis Desain Solusi Algoritma BFS dan DFS yang diimplementasikan

Dari hasil pengujian, dapat disimpulkan secara umum bahwa DFS lebih baik dalam mencari single file dan BFS lebih baik dalam mencari file yang banyak. Hal Ini didukung oleh data time spent yang terdapat pada setiap gambar. Pencarian BFS untuk kasus pertama menghasilkan 6 ms yang lebih lambat dari DFS yang hanya memerlukan waktu 4 ms. Sedangkan pencarian find all BFS pada kasus pertama menghasilkan waktu yang tidak jauh berbeda untuk single file yaitu 8 ms dibandingkan find all DFS yang memerlukan waktu yang lebih lama yaitu 12 ms. Hal ini didukung juga oleh kasus kedua yang memerlukan waktu 14 ms, 9 ms, 16 ms, dan 28 ms untuk sekuens pencarian yang sama.

Pada pengujian ketiga digunakan direktori yang nyata ditemukan dalam file system penguji. Hasil pencarian BFS dan DFS mirip karena kedalaman rendah dan sekuens pencarian BFS dan DFS untuk file system tersebut sama. Untuk Findall juga serupa untuk kedua kasus sehingga dapat disimpulkan pada kasus biasa, tidak ada perbedaan signifikan antara BFS dan DFS.

Algoritma juga memanfaatkan urutan pembangkitan mulai dari file baru masuk ke folder untuk optimasi heuristik karena algoritma DFS akan lebih lama jika pembangkitan file dan folder disatukan.

Aplikasi tidak bisa melakukan visualisasi untuk direktori yang memiliki banyak file dengan nama yang panjang karena *library* MSAGL yang dipakai untuk menampilkan pohon menampilkan cabang dari suatu node pada garis yang sama sehingga graf yang dikeluarkan akan sangat lebar dan seringkali tidak bisa dibedakan dengan garis. Untuk kasus direktori seperti ini secara umum lebih menguntungkan memakai BFS karena semua file pada kedalaman yang serupa.

BAB 5
Kesimpulan dan Saran

5.1 Kesimpulan

Algoritma BFS dan DFS dapat digunakan untuk menyelesaikan permasalahan pencarian pada graf yaitu dengan cara traversal. BFS atau *Breadth First Search* merupakan pencarian yang dilakukan secara melebar, sedangkan DFS atau *Depth First Search* merupakan pencarian yang dilakukan secara mendalam.

Salah satu penerapan algoritma BFS dan DFS adalah untuk mencari file yang berada di dalam suatu direktori yang direpresentasikan sebagai suatu pohon. Algoritma BFS dan DFS akan memberikan hasil pencarian yang berbeda dengan waktu pencarian yang berbeda pula, bergantung pada kasus yang sedang ditelusuri.

5.2 Saran

Hasil program yang telah dibuat masih bisa diperbaiki dan dikembangkan lebih jauh. Di antaranya yaitu:

1. Memperbaiki tampilan pohon filesystem agar lebih jelas untuk direktori yang mengandung banyak file dan upadirektori.
2. Menampilkan proses pencarian sebagai pohon yang terus bertambah simpul-simpulnya sesuai simpul yang sedang dan sudah ditinjau, baik untuk BFS maupun DFS.
3. Memakai Contoh Kasus dengan File Solusi yang memiliki kedalaman berbeda-beda

Link Github : https://github.com/nart4hire/Tubes_2_Stima.git

Link Youtube : https://youtu.be/prCglaCF-I8

Daftar Pustaka

Dirk Riehle. Framework Design: A Role Modeling Approach. Ph.D. Thesis, No. 13509. Zürich, Switzerland, ETH Zürich, 2000.

Filus, Teo. *Pengenalan Bahasa Pemrograman C#*. 18 Januari 2017. https://codepolitan.com/pengenalan-bahasa-pemrograman-c-587effa1cb95b/.

Foundation, .NET (n.d.). *What is .net? an open-source developer platform.* Microsoft. Retrieved March 25, 2022, from https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet.

Munir, Rinaldi, dan Nur Ulfa Maulidevi. 2021. *Breadth/Depth First Search (BFS/DFS)*. Bandung: Program Studi Teknik Elektro dan Informatika ITB.

Wagner, B. (n.d.). *C# docs - get started, tutorials, reference.* C# docs - get started, tutorials, reference. | Microsoft Docs. Retrieved March 25, 2022, from https://docs.microsoft.com/en-us/dotnet/csharp/