

**TUGAS BESAR II**  
**PENGAPLIKASIAN ALGORITMA BFS DAN DFS**  
**DALAM IMPLEMENTASI FOLDER CRAWLING**  
**IF2211 – STRATEGI ALGORITMA**



**Disusun oleh:**

**Kelompok 10**

**BreadFS**

13520034	Bryan Bernigen
13520112	Fernaldy
13520166	Raden Rifqi Rahman

**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**Jl. Ganesha No. 10, Bandung 40132**

## Daftar Isi

Bab 1 Deskripsi Tugas .....	1
Bab 2 Landasan Teori .....	4
2.1 Traversal Graf.....	4
2.2 DFS .....	4
2.3 BFS.....	5
2.4 C# Desktop application development.....	6
Bab 3 Analisis Penyelesaian Masalah.....	7
3.1 Langkah-langkah Pemecahan Masalah .....	7
3.2 <i>Mapping</i> Persoalan .....	7
3.3 Contoh Ilustrasi Kasus lain .....	8
Bab 4 Implementasi dan Pengujian.....	11
4.1 Implementasi program .....	11
4.2 Penjelasan Struktur Data .....	12
4.3 Cara Penggunaan Program .....	14
4.4 Hasil Pengujian.....	15
4.5 Analisis dari desain Solusi BFS dan DFS.....	17
Bab 5 Kesimpulan dan Saran .....	20
Daftar Pustaka.....	21
Lampiran.....	22

## Bab 1 Deskripsi Tugas

Pada Tugas Kali ini, Mahasiswa diminta untuk mengimplementasikan ilmu mengenai BFS dan DFS yang mereka dapatkan di kelas. Mahasiswa diminta untuk membuat sebuah aplikasi folder crawling berbasis graphical user interface (GUI) untuk mencari sebuah file dalam sebuah folder. Program akan meminta dua buah input yakni folder awal tempat file yang akan dicari dan nama file yang akan dicari. Lalu program akan meminta user untuk memilih metode yang dipakai untuk mencari file yakni BFS atau DFS. Program juga akan meminta user untuk memilih apakah program hanya akan mencari satu buah file atau seluruh file. Setelah itu program akan menampilkan hasil pencariannya dalam bentuk pohon dan jika file ketemu, program akan menampilkan *hyperlink* menuju file tersebut. Berikut merupakan spesifikasi program secara lengkap.

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 hasil pencarian file tersebut dengan memberikan keterangan folder/file yang sudah diperiksa, folder/file yang sudah masuk antrian tapi belum diperiksa, dan rute folder serta 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 hasil 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** (<https://github.com/microsoft/automatic-graph-layout>) Berikut ini adalah panduan singkat terkait penggunaan MSAGL oleh tim asisten yang dapat diakses pada:  
<https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVI3MY6gt1t-PL30LA/edit?usp=sharing>
- 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 2 Landasan Teori

### 2.1 Traversal Graf

Traversal graf adalah sebuah algoritma yang mengunjungi simpul-simpul sebuah graf secara sistematis dan terstruktur untuk mencari solusi dari sebuah masalah yang direpresentasikan dalam bentuk graf. Secara umum, terdapat dua pendekatan dalam pencarian solusi pada traversal graf yakni traversal graf tanpa informasi atau traversal informasi dengan informasi. Traversal graf tanpa informasi memanfaatkan *brute force* dalam penerapannya sedangkan traversal graf dengan informasi memanfaatkan heuristik dalam penerapannya. Salah satu contoh dari algoritma traversal graf tanpa informasi adalah algoritma *depth first search*(DFS) dan algoritma *breadth first search*(BFS).

Dalam pencarian solusi menggunakan traversal graf, tentu saja terdapat graf yang menjadi representasi permasalahan. Secara umum representasi graf tersebut dapat dibagi menjadi dua, yakni graf statis dan graf dinamis. Graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dilakukan. Graf statis biasanya digunakan untuk merepresentasikan masalah yang lingkungannya terbatas. Salah satu contoh graf statis adalah pencarian file dalam sebuah direktori. Pada contoh ini, lingkup permasalahan terbatas pada direktori yang ada sehingga permasalahan tersebut dapat direpresentasikan sebagai graf statis. Graf dinamis adalah graf yang belum terbentuk ketika pencarian dilakukan, namun graf tersebut terbentuk secara bersamaan ketika algoritma sedang mencari solusi. Graf dinamis biasanya digunakan untuk merepresentasikan permasalahan dengan lingkup yang tidak terbatas. Salah satu contoh graf dinamis adalah pencarian solusi dalam permainan catur. Pada permasalahan tersebut, lingkup masalah tidaklah terbatas karena jumlah pergerakan untuk menyelesaikan permainan dapat mencapai tak hingga (jika raja hanya keliling maju, kanan, mundur, kiri). Oleh karena itu, masalah ini direpresentasikan sebagai graf dinamis.

### 2.2 DFS

*Depth First Search* (DFS) adalah salah satu algoritma yang dapat digunakan untuk melakukan pencarian secara traversal pada sebuah graf. DFS menelusuri graf secara mendalam yang berarti algoritma DFS akan memproses anak sebuah simpul terlebih dahulu sebelum saudara dari simpul tersebut diproses (Contoh: Algoritma DFS akan memproses 1.1.1 sebelum memproses 1.2). Algoritma DFS biasanya direpresentasikan sebagai sebuah rekursi karena

sifatnya yang terus maju lalu *backtrack* ketika sudah berada di *level* terdalam atau ketika seluruh anak simpul tersebut sudah dikunjungi.

Berikut merupakan cara penerapan algoritma DFS secara umum:

1. Kunjungi sebuah simpul x sebagai simpul pertama.
2. Proses simpul x.
3. Jika x sudah sesuai, pencarian dapat dihentikan.  
Jika x belum sesuai, kunjungi simpul y yang merupakan anak dari x dan belum pernah di proses. Ulangi langkah 1-3 dengan simpul y sebagai simpul pertama.
4. Jika seluruh anak dari x sudah pernah di proses, maka lakukan *backtrack* pada simpul z yang dikunjungi sebelum simpul x. Lakukan langkah 3 pada simpul z tersebut.
5. Jika tidak ada simpul yang dikunjungi sebelum simpul x dan tidak ada lagi anak dari simpul x yang belum pernah di kunjungi, maka pencarian berakhir dengan kesimpulan bahwa hal yang dicari tidak ditemukan pada graf tersebut.

Keunggulan dari algoritma DFS adalah mampu menemukan simpul dengan lebih cepat apabila simpul tersebut terletak jauh dari akar. Sementara itu, kelemahan algoritma DFS adalah adanya kemungkinan menemukan solusi yang tidak optimal.

## 2.3 BFS

*Breadth First Search* (BFS) adalah salah satu algoritma yang dapat digunakan untuk melakukan pencarian secara traversal pada sebuah graf. BFS menelusuri graf secara melebar yang berarti algoritma BFS akan memproses saudara sebuah simpul terlebih dahulu sebelum memproses anak dari simpul tersebut (Contoh: Algoritma BFS akan memproses 1.1, 1.2, 1.3 terlebih dahulu sebelum memproses 1.1.1). Algoritma BFS biasanya direpresentasikan dengan sebuah antrean karena sifatnya yang akan mengecek seluruh simpul pada sebuah kedalaman terlebih dahulu sebelum maju ke kedalaman selanjutnya.

Berikut Merupakan cara penerapan algoritma BFS secara umum:

1. Siapkan sebuah antrean kosong dan sebuah catatan yang mencatat apakah sebuah simpul sudah diproses atau belum.
2. Kunjungi sebuah simpul x sebagai simpul pertama.
3. Proses simpul x dan catat bahwa simpul x sudah di proses.
4. Jika x sudah sesuai, pencarian dapat dihentikan.  
Jika x belum sesuai, masukan semua simpul yang bertetangga dengan x ke dalam antrean.

5. Pilih salah satu simpul yang berada pada antrean dan hapus simpul tersebut dari antrean. Jika simpul tersebut sudah pernah di proses, ulangi langkah 5 selama antrean belum kosong. Jika simpul belum pernah di proses, ulangi langkah 2 dengan simpul yang dipilih sebagai simpul pertama.
6. Jika antrean kosong, maka dapat disimpulkan bahwa hal yang dicari tidak ditemukan di graf tersebut.

Kelebihan algoritma BFS adalah selalu menemukan solusi yang optimal. Sementara itu, kelemahan algoritma BFS adalah memakan memori lebih banyak daripada algoritma DFS.

## **2.4 C# Desktop application development**

C# adalah bahasa pemrograman berorientasi objek yang dikembangkan oleh Microsoft. Bahasa pemrograman C# dapat digunakan untuk pengembangan berbagai jenis aplikasi, seperti aplikasi *web*, aplikasi *mobile*, aplikasi *console*, dan berbagai jenis aplikasi lainnya. Program yang dibuat dalam bahasa C# dapat dijalankan pada .NET yaitu sebuah *virtual execution system* dan sekumpulan *class libraries*. Pemrograman menggunakan bahasa C# dapat memanfaatkan Microsoft Visual Studio, sebuah *integrated development environment* (IDE) yang dikembangkan oleh Microsoft.



## Bab 3 Analisis Penyelesaian Masalah

### 3.1 Langkah-langkah Pemecahan Masalah

Berikut adalah langkah-langkah pemecahan masalah dengan algoritma DFS:

1. Menentukan direktori awal pencarian *file* dan nama *file* yang akan dicari.
2. Menelusuri setiap *file* yang ada dalam direktori awal dan mencocokkannya dengan *file* yang dicari.
3. Apabila *file* yang dicari tidak ada dalam direktori yang sedang ditelusuri atau ingin menemukan semua *file* yang cocok dalam direktori awal, lakukan penelusuran setiap *folder* yang ada dalam direktori.
4. Lakukan pemanggilan rekursif algoritma DFS terhadap setiap *folder* tersebut.

Berikut adalah langkah-langkah pemecahan masalah dengan algoritma BFS:

1. Menentukan direktori awal pencarian *file* dan nama *file* yang akan dicari.
2. Membuat suatu antrean kosong yang akan berisi direktori-direktori yang telah dikunjungi dan memasukkan direktori awal ke dalam antrean tersebut.
3. Selama antrean belum kosong, lakukan *dequeue* untuk memperoleh suatu direktori pada antrean tersebut. Lakukan penelusuran terhadap setiap *file* yang ada dalam direktori tersebut dan mencocokkannya dengan *file* yang dicari. Apabila *file* yang dicari tidak ada dalam direktori yang sedang ditelusuri atau ingin menemukan semua *file* yang cocok dalam direktori awal, lakukan penelusuran setiap *folder* yang ada dalam direktori. Setiap *folder* tersebut di-*enqueue* ke dalam antrean.

### 3.2 Mapping Persoalan

Mapping persoalan pada DFS:

1. Larik boolean “dikunjungi” adalah larik yang bernilai *true* untuk semua *directory* yang telah ditelusuri dan semua *file* yang telah dicocokkan dengan *file* yang dicari serta *false* untuk semua *directory* dan *file* yang belum dikunjungi.
2. Matriks ketetanggaan  $A = [a_{ij}]$  adalah matriks yang  $a_{ij}$  bernilai 1 jika *file* atau *folder*  $j$  ada dalam *folder*  $i$  dan bernilai 0 jika *file* atau *folder*  $j$  tidak ada dalam *folder*  $i$ .

Mapping persoalan pada BFS:

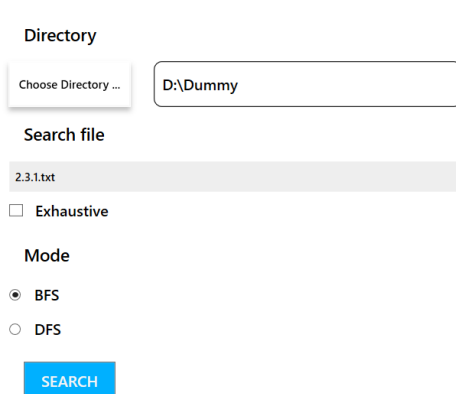
1. Larik boolean “dikunjungi” adalah larik yang bernilai *true* untuk semua *directory* yang telah ditelusuri dan semua *file* yang telah dicocokkan dengan *file* yang dicari serta *false* untuk semua *directory* dan *file* yang belum dikunjungi.
2. Matriks ketetanggaan  $A = [a_{ij}]$  adalah matriks yang  $a_{ij}$  bernilai 1 jika *file* atau *folder*  $j$  ada dalam *folder*  $i$  dan bernilai 0 jika *file* atau *folder*  $j$  tidak ada dalam *folder*  $i$ .
3. Antrean  $q$  untuk menyimpan *directory* *folder* yang telah dikunjungi dan belum ditelusuri.

### 3.3 Contoh Ilustrasi Kasus lain

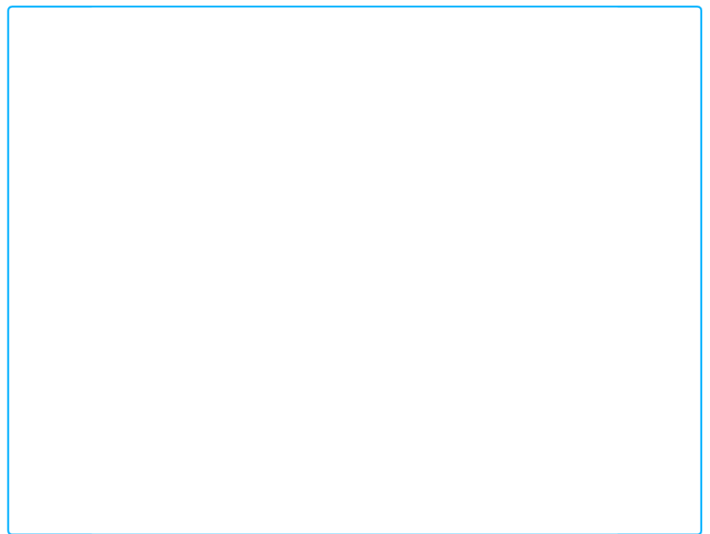
Contoh kasus 1:

- a) Contoh masukan:

#### Folder Crawler



The screenshot shows the 'Folder Crawler' application interface. It features a 'Directory' section with a 'Choose Directory ...' button and a text input field containing 'D:\Dummy'. Below this is a 'Search file' section with a text input field containing '2.3.1.txt'. There is an unchecked checkbox labeled 'Exhaustive'. The 'Mode' section has two radio buttons: 'BFS' (which is selected) and 'DFS'. At the bottom is a blue 'SEARCH' button.



b) Contoh keluaran:

## Folder Crawler

Directory

Choose Directory ...

D:\Dummy

Search file

2.3.1.txt

☐ Exhaustive

Mode

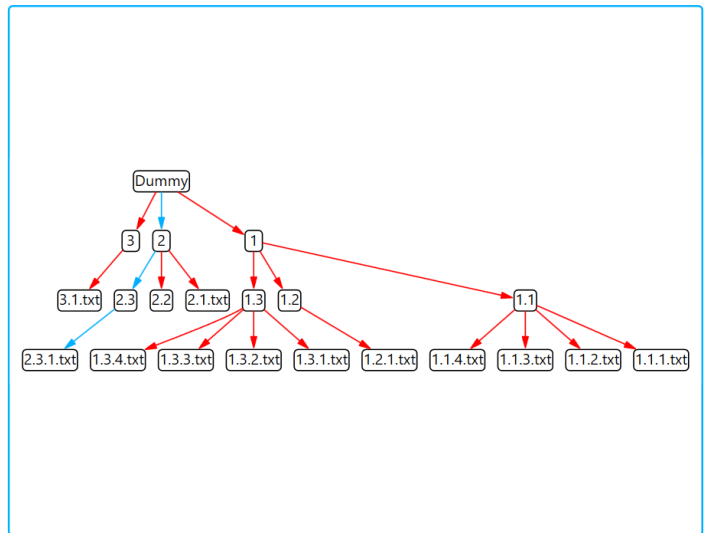
☒ BFS

☐ DFS

SEARCH

D:\Dummy\2\2.3\2.3.1.txt

Waktu Pencarian: 871 ms



Pada kasus 1, pengguna ingin mencari *file* 2.3.1.txt pada direktori D:\Dummy.

*Path* pencarian menggunakan algoritma BFS adalah D:\Dummy → 1 → 2 → 3 → 1.1 → 1.2 → 1.3 → 2.1.txt → 2.2 → 2.3 → 3.1.txt → 1.1.1.txt → 1.1.2.txt → 1.1.3.txt → 1.1.4.txt → 1.2.1.txt → 1.3.1.txt → 1.3.2.txt → 1.3.3.txt → 1.3.4.txt → 2.3.1.txt

Pada keluaran, sisi berwarna merah menandakan bahwa simpul tersebut merupakan *folder* yang sudah ditelusuri dan tidak berisi *file* yang dicari atau *file* yang sudah dicocokkan dengan *file* yang dicari dan tidak sesuai. Sisi berwarna biru menandakan rute dari direktori awal ke *file* yang dicari. Sementara itu, sisi berwarna hitam tidak ada karena tidak ada *folder* yang berada dalam antrian pencarian dan belum ditelusuri.

Contoh kasus 2:

a) Contoh masukan:

### Folder Crawler

Directory

Choose Directory ... D:\Dummy\1

Search file

2.3.1.txt

☐ Exhaustive

Mode

☐ BFS

☒ DFS

SEARCH



b) Contoh keluaran:

### Folder Crawler

Directory

Choose Directory ... D:\Dummy\1

Search file

2.3.1.txt

☐ Exhaustive

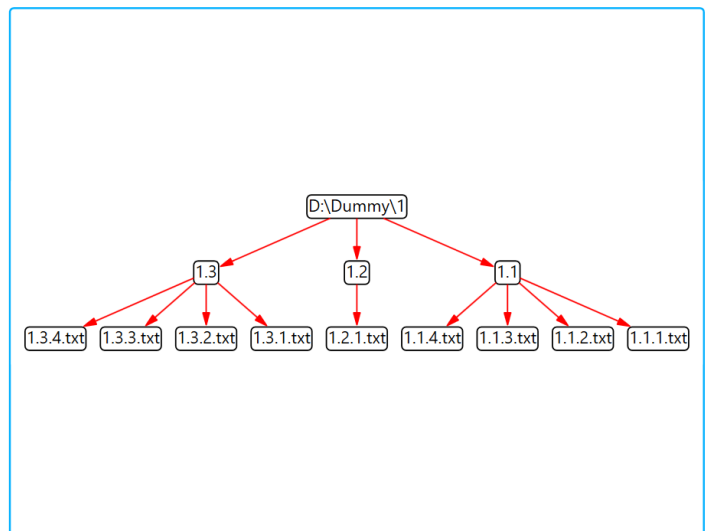
Mode

☐ BFS

☒ DFS

SEARCH

File not found!



Pada kasus 2, pengguna ingin mencari file 2.3.1.txt pada direktori D:\Dummy\1.

*Path* pencarian menggunakan algoritma BFS adalah D:\Dummy\1 → 1.1 → 1.1.1.txt → 1.1 → 1.1.2.txt → 1.1 → 1.1.3.txt → 1.1 → 1.1.4.txt → 1.1 → D:\Dummy\1 → 1.2 → 1.2.1.txt → 1.2 → D:\Dummy\1 → 1.3 → 1.3.1.txt → 1.3 → 1.3.2.txt → 1.3 → 1.3.3.txt → 1.3 → 1.3.4.txt → 1.3 → D:\Dummy\1

Pada keluaran, semua sisi berwarna merah karena file 2.3.1.txt tidak ditemukan pada direktori D:\Dummy\1.

## Bab 4 Implementasi dan Pengujian

### 4.1 Implementasi program

Program Utama:

```
input(directory)
input(namafile)
input(exhaustive)
input(useDfs, useBfs)
valid  $\leftarrow$  Validasi(directory, namafile, exhaustive, useDfs, useBfs)
if (valid) then
    if (useDfs) then
        DFS(directory, namafile, exhaustive)
    else
        BFS(directory, namafile, exhaustive)
```

DFS( Path\_folder, file\_yang\_dicari : string, semua\_file : bool):

```
If (file belum ketemu V mencari semua file) then
    files  $\leftarrow$  nama-nama file di folder tersebut
    While (masih ada file ke-i dalam files yang belum ditelusuri) do
        tambah simpul file ke-i ke graf
        tambah edge dari Path_folder ke file ke-i
        If (file ke-i = file yang dicari) then
            Ketemu  $\leftarrow$  true
    If(file belum ketemu V tipe mencari semua file) then
        folders  $\leftarrow$  nama-nama folder yang ada di Path_folder tersebut
        While(masih ada folder ke-i dalam folders yang belum ditelusuri) do
            tambah simpul folder ke-i ke graf
            tambah edge dari Path_folder ke folder ke-i
            DFS(folder ke-i, file yang dicari, semua file)
```

Warnai semua path yang menjadi solusi

BFS(Path\_folder, file\_yang\_dicari: string, semua\_file: bool):

```
BuatAntrian(q)
enqueue(q, Path_folder)
```

```

while (q tidak kosong  $\wedge$  (belum ketemu V mencari semua file)) do
    dequeue(q, current)
    tambah node current ke graf
    files  $\leftarrow$  nama-nama file di current tersebut
    while (masih ada files  $\wedge$  (belum ketemu V mencari semua file)) do
        tambah node file ke-i ke graf
        tambah edge dari current ke file ke-i
        If (files ke-i = file yang dicari) then
            ketemu  $\leftarrow$  true
    folders  $\leftarrow$  nama-nama folder di current tersebut
    while (masih ada folder  $\wedge$  (belum ketemu V mencari semua file)) do
        tambah node folder ke-i ke graf
        tambah edge dari current ke folder ke-i
        enqueue(q, folder)
Warnai semua path yang menjadi solusi

```

## 4.2 Penjelasan Struktur Data

Berikut adalah struktur data yang digunakan:

### 1. Kelas GraphContext

Atribut kelas GraphContext terdiri dari `_control`, `_nullGraph`, `_nodes`, `_edges`, dan `_parents`. Atribut `_nodes` merupakan *dictionary* yang menyimpan semua informasi path dan simpul path tersebut. Atribut `_edges` merupakan *dictionary* yang menyimpan semua informasi path anak dan sisi yang berkaitan. Atribut `parents` merupakan *dictionary* yang menyimpan semua informasi path anak dan path orang tua. *Method* kelas GraphContext adalah `ResetGraph` untuk melakukan reset graf, `AddNode` untuk menambahkan simpul baru, `AddEdge` untuk menambahkan sisi baru, `ColorizeEdge` untuk mewarnai sebuah sisi, `ColorizePath` untuk mewarnai semua sisi dari suatu simpul ke akar, dan `UpdateView` untuk memperbaharui tampilan graf.

### 2. Kelas ColorPalette

Atribut kelas ColorPalette yaitu `RED` dan `BLUE` yang masing-masing merepresentasikan warna merah dan biru.

### 3. Kelas Bfs

Atribut kelas Bfs adalah `_graphContext` yaitu instantiasi dari kelas `GraphContext` dan `_result` yang menyimpan semua *path* solusi. *Method* `Search` merupakan implementasi dari algoritma BFS.

#### 4. Kelas Dfs

Atribut kelas Dfs adalah `_found` yang menunjukkan apakah solusi sudah ditemukan, `_graphContext` yaitu instantiasi dari kelas `GraphContext`, dan `_result` yang menyimpan semua *path* solusi. *Method* `Search` merupakan *method* yang dipanggil untuk melakukan DFS. Sementara itu, *method* `RecursiveSearch` adalah implementasi algoritma DFS yang akan dipanggil secara rekursif.

#### 5. Kelas MainWindow

Atribut kelas `MainWindow` adalah `_graphContext` yang merupakan instantiasi dari kelas `GraphContext`. *Method* `OnChooseDir` digunakan untuk menerima *directory* yang akan dilakukan penelusuran. *Method* `OnSearch` digunakan untuk memulai pencarian. *Method* `ValidateInput` digunakan untuk memvalidasi input. *Method* `HideErrorMsg` digunakan untuk menyembunyikan pesan error. *Method* `ShowSearchResult` digunakan untuk menampilkan kan pencarian. *Method* `CreateNewResultItem` digunakan untuk menghasilkan *hyperlink* ke *directory* hasil.

Berikut adalah spesifikasi umum program:

1. Program dapat menerima input direktori folder dan nama file yang dicari.
2. Program dapat menerima pilihan mencari semua file yang sesuai atau tidak.
3. Program dapat menerima pilihan algoritma apa yang ingin digunakan BFS atau DFS
4. Program dapat menampilkan pohon hasil pencarian secara *real-time* serta memberikan keterangan:
  - a. Sisi berwarna merah  
Folder yang sudah ditelusuri dan tidak berisi file yang dicari atau file yang tidak sesuai dengan yang dicari
  - b. Sisi berwarna hitam  
Folder yang sudah masuk antrean dan belum ditelusuri atau file yang masuk antrean dan belum dicocokkan dengan file yang dicari
  - c. Sisi berwarna biru  
Folder yang merupakan direktori tempat file ditemukan atau file merupakan yang dicari

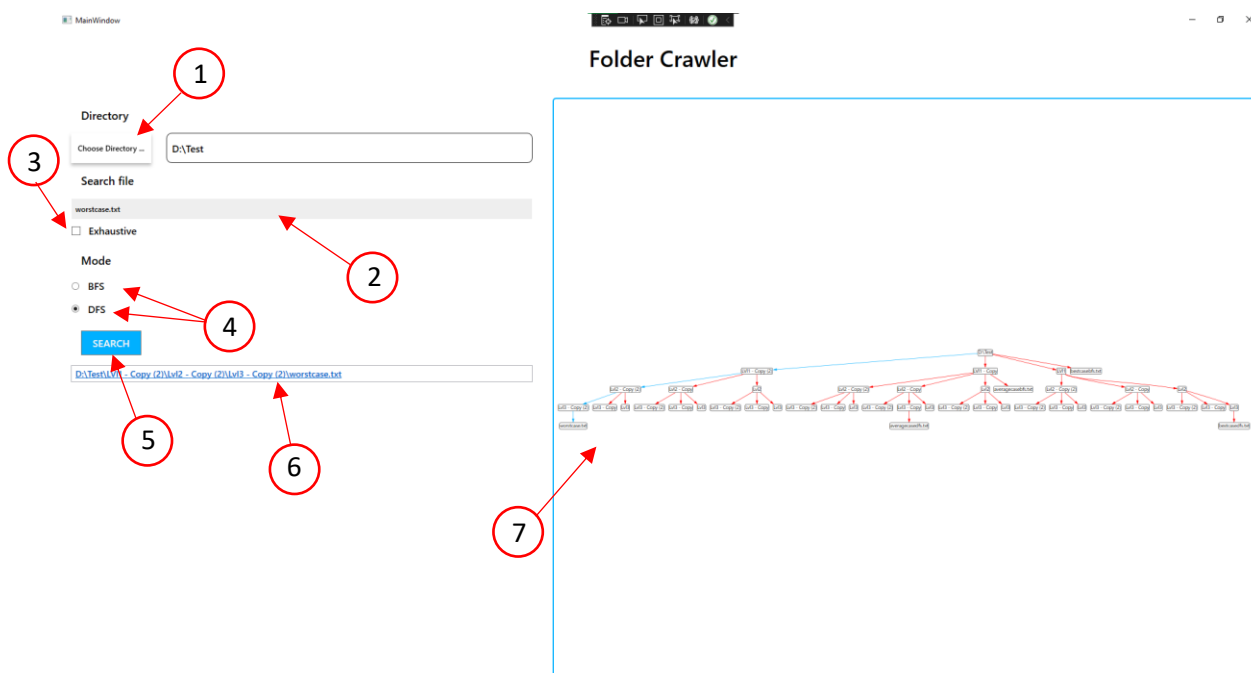
5. Program dapat menghasilkan hyperlink menuju folder tempat file yang dicari ditemukan. File explorer akan terbuka jika hyperlink tersebut diklik.

### 4.3 Cara Penggunaan Program

Program dibuat dalam bahasa C# dan memanfaatkan kakas Visual Studio .NET untuk mempermudah pengembangan. Berikut adalah langkah-langkah penggunaan program:

1. Lakukan instalasi Visual Studio .NET yang dapat diunduh dari <https://visualstudio.microsoft.com/downloads/>
2. Lakukan instalasi NuGet Package untuk MSAGL dengan cara menggunakan *command* berikut pada Package Manager Console:  
Install-Package AutomaticGraphLayout -Version 1.1.11  
Install-Package AutomaticGraphLayout.Drawing -Version 1.1.11  
Install-Package AutomaticGraphLayout.WpfGraphControl -Version 1.1.11  
Install-Package AutomaticGraphLayout.GraphViewerGDI -Version 1.1.11
3. Buka *project* dengan nama Tubes2\_13520034.sln dan lakukan *start debugging*

Berikut adalah tampilan aplikasi ketika program dijalankan:



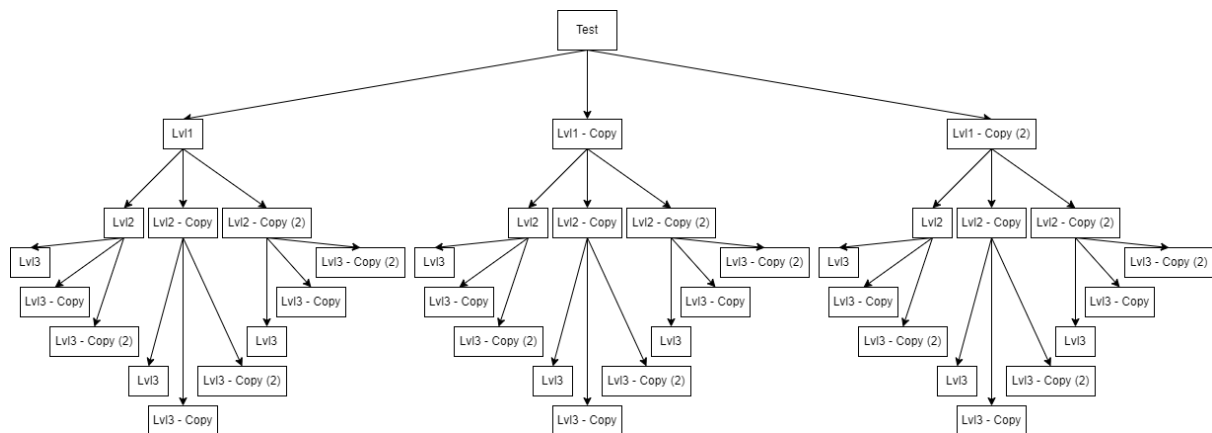


- 1) Tombol untuk memilih direktori awal tempat pencarian dilakukan.
- 2) *Box input* untuk memasukkan nama *file* yang akan dicari (contoh: a.txt, b.docx, c.jpg).
- 3) Tombol untuk memilih apakah program akan mencari seluruh *file* atau hanya satu *file* saja. Jika *box* dicentang, maka program akan mencari seluruh *file* pada direktori tersebut.
- 4) Tombol untuk memilih metode pencarian yakni BFS atau DFS.
- 5) Tombol untuk mulai melakukan pencarian.
- 6) *Hyperlink* yang akan muncul ketika *file* yang dicari ditemukan dalam direktori. Pengguna dapat menekan tulisan tersebut untuk membuka lokasi *file* tersebut di *file explorer*.
- 7) Tampilan graf pencarian yang selalu *ter-update* selama pencarian. Graf tersebut dapat diperbesar ataupun diperkecil dengan menggunakan *scroll* pada *mouse*.

#### 4.4 Hasil Pengujian

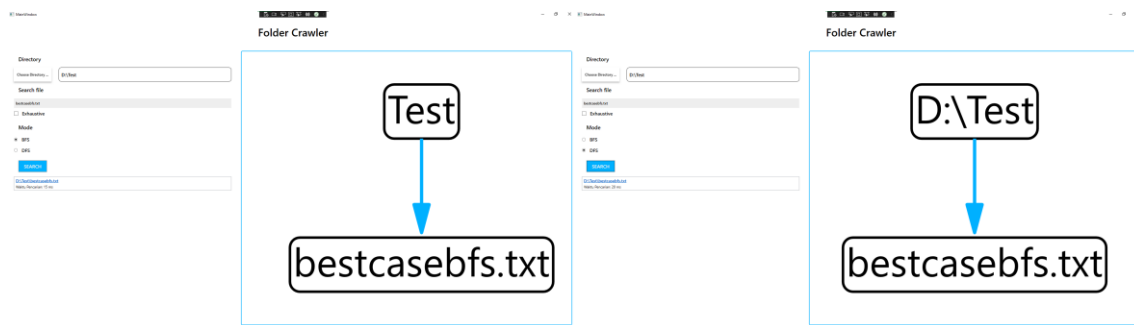
Struktur folder yang akan diujikan (Tanpa File):

Folder test memiliki 3 buah folder yakni Lvl1, Lvl1 – Copy, Lvl1 – Copy (2). Setiap folder tersebut juga memiliki 3 buah folder Lvl2, Lvl2 – Copy, Lvl2 – Copy(2). Setiap folder Lvl2 memiliki 3 buah folder yakni Lvl3, Lvl3 – Copy, dan Lvl3 – Copy (2).



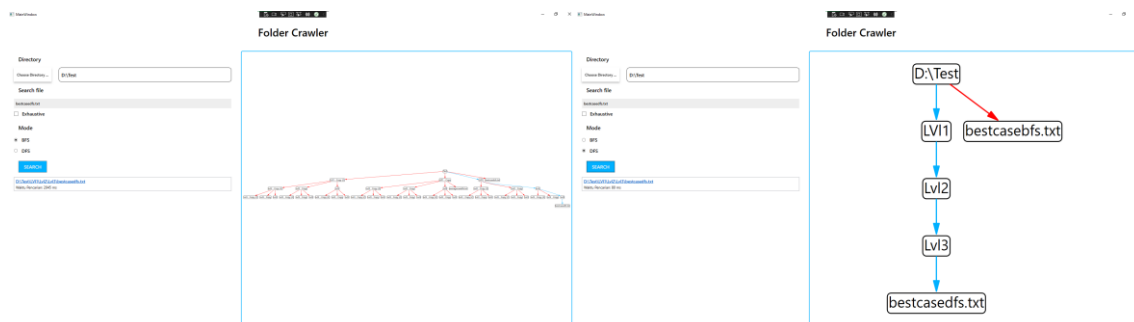
Gambar Struktur Folder Testing

### Testcase 1: penambahan bestcasebfs.txt pada folder test



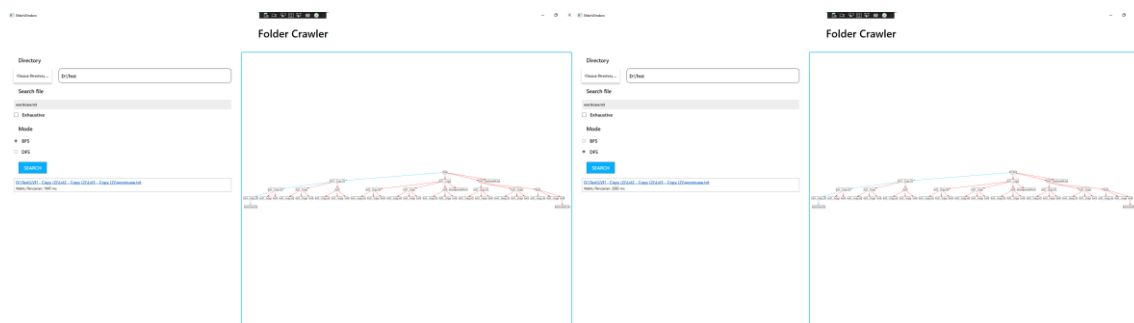
Pencarian bestcasedfs.txt dengan BFS(kiri) dan DFS(Kanan)

### Testcase 2: Penambahan bestcasedfs.txt pada Folder Test/Lvl1/Lvl2/Lvl3



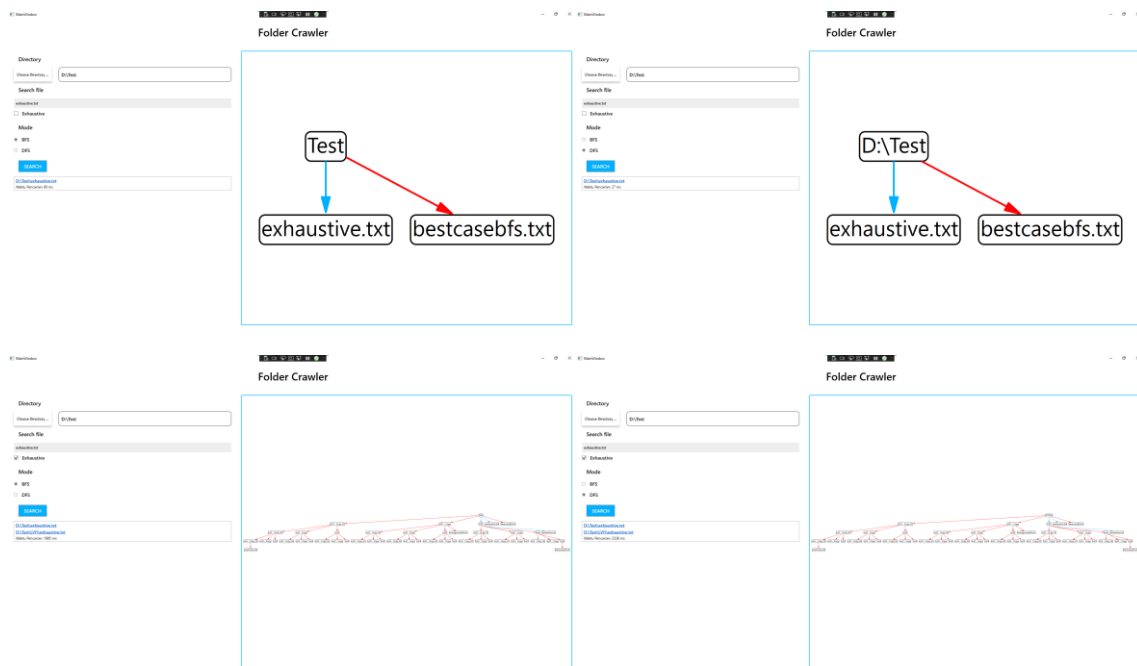
Pencarian bestcasedfs.txt dengan BFS(kiri) dan DFS(kanan)

### Testcase 3: Penambahan worstcase.txt pada Test/Lvl1 – Copy (2)/ Lvl2 – Copy (2)/ Lvl3 – Copy (2)



Pencarian Worstcase.txt dengan BFS(kiri) dan DFS(kanan)

#### Testcase 4: Penambahan exhaustive.txt pada folder Test dan Test/Lvl1



Pencarian exhaustive.txt dengan BFS(kiri) dan DFS(kanan) dengan pilihan Exhaustive pada program tidak tercentang(atas) dan tercentang(bawah)

### 4.5 Analisis dari desain Solusi BFS dan DFS

#### Skenario 1: File yang dicari berada pada direktori awal

Kasus	BFS	DFS	Hasil
Terbaik: File berada pada urutan ke-1 di direktori tersebut sehingga langsung ketemu.	File langsung ketemu	File langsung ketemu	Seri
Rata-rata: File berada di antara urutan ke-1 dan urutan ke-n	Program mengecek antara 1 sampai (n-1) item lainnya tanpa menelusuri folder sebelum file ketemu	Program mengecek secara rekursif semua file dan folder yang terdapat pada direktori awal sebelum item ke-i dengan i antara 1 sampai n	BFS
Terburuk: File yang dicari berada	Program mengecek (n-1) buah item tanpa	Program mengecek secara rekursif semua file dan folder yang terdapat	BFS

pada urutan terakhir di direktori	menelusuri folder sebelum file ketemu	pada direktori awal sebelum item ke-n	
-----------------------------------	---------------------------------------	---------------------------------------	--

Pada skenario tersebut, BFS jauh lebih baik dibandingkan DFS karena pada skenario terburuknya, BFS hanya akan mengecek  $n$  buah file dengan  $n$  banyaknya file/folder tanpa menelusuri folder dalam direktori utama sebelum akhirnya menemukan file yang dicari. Berbeda dengan DFS yang pada kasus terburuknya mungkin harus mengecek hampir seluruh file pada direktori.

### Skenario 2: File berada pada direktori terdalam

Kasus	BFS	DFS	Hasil
Terbaik: File berada pada direktori terdalam dari folder-folder pertama yang di cek	Program mengecek seluruh file/folder setiap kedalaman sampai kedalaman terakhir dan pada kedalaman terakhir langsung menemukan file yang dicari.	Program hanya menelusuri 1 folder tiap kedalaman hingga kedalaman terakhir dan menemukan file yang dicari	DFS
Rata-rata: File berada pada direktori terdalam dan file tidak berada pada folder terakhir yang ditelusuri	Program mengecek seluruh file/folder pada setiap kedalaman sampai kedalaman terakhir dan pada kedalaman terakhir program mengecek $n$ item sebelum menemukan file yang dicari dengan $n$ adalah 0 sampai $(n-1)$ item pada kedalaman terakhir	Program mengecek seluruh file/folder dalam setiap direktori yang mungkin ditemukan secara rekursif sebelum direktori file yang dicari.	DFS
Terburuk: File berada pada direktori terakhir	Program mengecek semua file/folder sebelum file ditemukan	Program mengecek semua file/folder secara rekursif sebelum file ditemukan	Seri

Pada skenario tersebut, DFS lebih baik karena file baru dapat ditemukan pada kedalaman terdalam yang berarti BFS akan menghabiskan waktunya ketika mencari file pada kedalaman-

kedalaman yang belum terdalam. Namun pada kondisi terburuk, DFS akan sama dengan BFS pada skenario tersebut.

## Bab 5 Kesimpulan dan Saran

Kesimpulan yang diperoleh adalah BFS mengutamakan pencarian ke samping atau melebar sehingga algoritma tersebut lebih baik untuk digunakan ketika file yang ingin dicari berada pada kedalaman yang dangkal. Sementara itu, DFS mengutamakan kedalaman dalam proses penelusuran sehingga algoritma DFS lebih baik digunakan ketika file yang ingin dicari berada pada aras yang jauh dari akar.

Saran untuk pengembangan lanjutan adalah untuk menambahkan kreativitas dalam tampilan antarmuka aplikasi yang telah dibuat. Selain itu, bisa ditambahkan fitur pencarian terhadap folder, bukan hanya sebatas file. Sebagai tambahan, dapat diimplementasikan *stopwatch* selama pencarian agar waktu pencarian tidak hanya ditampilkan ketika pencarian selesai.

## 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/2021-2022/Tugas-Besar-2-IF2211-Strategi-Algoritma-2022.pdf>

<https://medium.com/@vatsalunadkat/advantages-and-disadvantages-of-ai-algorithms-d8fb137f4df2>

.

## **Lampiran**

Link Youtube: <https://youtu.be/HWAx0eMPo-8>