

LAPORAN TUGAS BESAR 2
IF2211 STRATEGI ALGORITMA



Disusun oleh:

Louis Yanggara	13520063
Jova Andres Riski Sirait	13520072
Dimas Faidh Muzaki	13520156

TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER 2 TAHUN 2021/2022

DAFTAR ISI

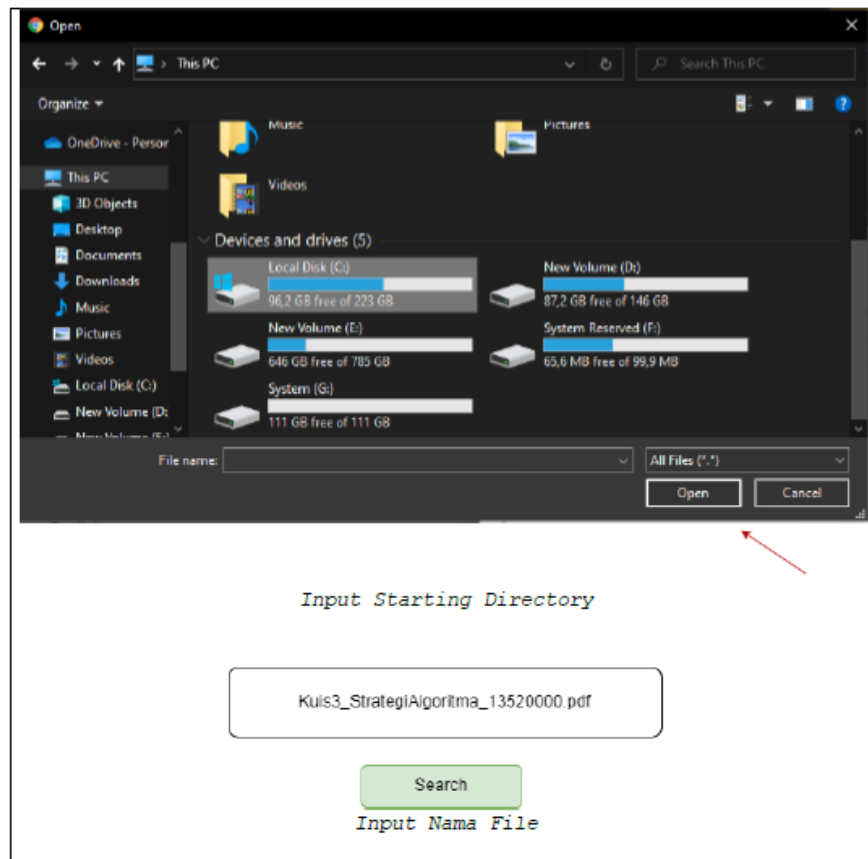
DAFTAR ISI	i
BAB I	1
BAB II	7
A. Graph Traversal.....	7
B. Breadth First Search.....	7
C. Depth First Search.....	7
D. C# Desktop Application Development.....	8
BAB III	11
BAB IV	14
A. Implementasi Program.....	14
B. Penjelasan Struktur Data	23
C. Cara Penggunaan Program	24
D. Hasil Pengujian.....	25
E. Analisis Desain Solusi Algoritma BFS dan DFS.....	29
BAB V	30
DAFTAR PUSTAKA	31

BAB I DESKRIPSI TUGAS

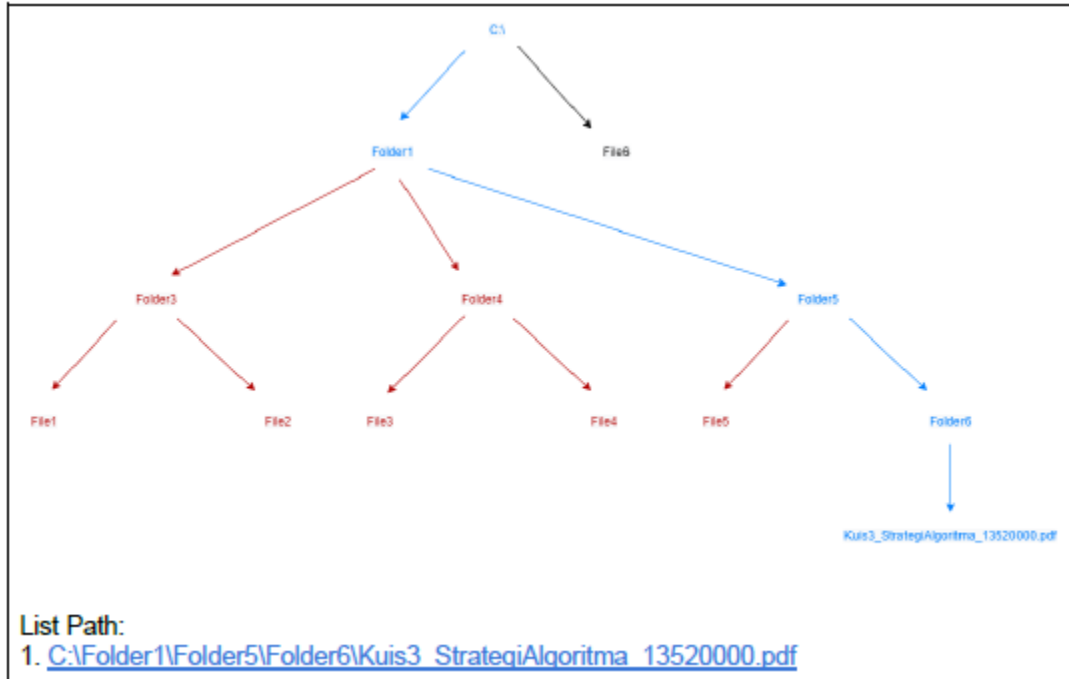
Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari file explorer pada sistem operasi, yang pada tugas ini disebut dengan Folder Crawling. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Anda dapat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

Contoh masukan aplikasi:



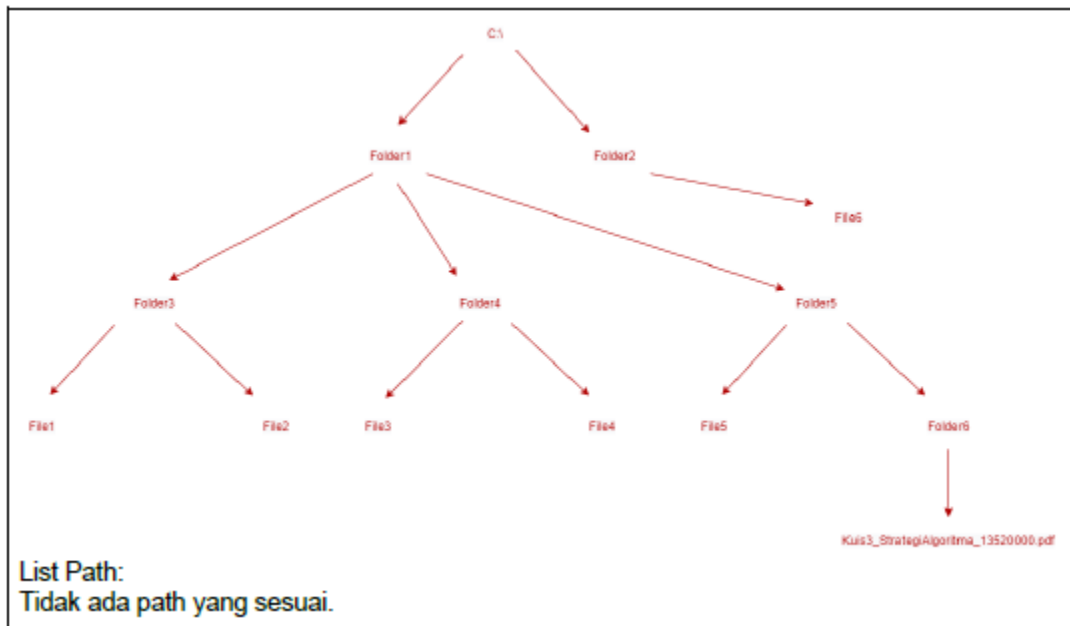
Contoh output aplikasi:



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.



Jika file yang ingin dicari pengguna tidak ada pada direktori file, misalnya saat pengguna mencari Kuis3Probststat.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.

Spesifikasi Program:

Aplikasi yang akan dibangun dibuat berbasis GUI. Berikut ini adalah contoh tampilan dari aplikasi GUI yang akan dibangun.

Folder Crawling

Input

Choose Starting Directory
[Choose Folder...](#) No File Chosen


Input File Name

☐ Find all occurrence

Input Metode Pencarian
☐ BFS
☒ DFS

[Search](#)

Output



Folder Crawling

Input

Choose Starting Directory
[Change Folder...](#) C:/

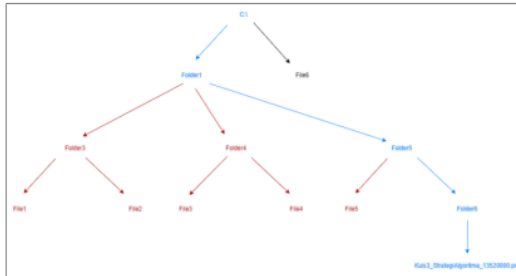
Input File Name

☐ Find all occurrence

Input Metode Pencarian
☐ BFS
☒ DFS

[Search](#)

Output



Path File :
• [C:/Folder1/Folder5/Folder6/Kuis3_StrategiAlgoritma_13520000.pdf](#)

Time spent: 20.02s

Catatan: Tampilan diatas hanya berupa salah satu contoh layout dari aplikasi saja, untuk design layout aplikasi dibebaskan dengan syarat mengandung seluruh input dan output yang terdapat pada spesifikasi.

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 kaskas yang tersedia. Sebagai referensi, salah satu kaskas 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 II

LANDASAN TEORI

A. Graph Traversal

Traversal graf merupakan sebuah teknik memproses graph dengan cara mengecek dan/atau memodifikasi vertex-vertex pada sebuah graph. Traversal graf dapat digunakan untuk menyelesaikan persoalan pencarian sebuah vertex pada sebuah graf. Persoalan tidak selamanya harus dalam bentuk graf, tetapi harus dapat dikonsepkan ke dalam struktur data graf ataupun pohon. Kelebihan dari traversal graf adalah dapat menghindari terjadinya loop pada proses traversal. Hal ini dapat dicapai dengan cara menandai simpul-simpul yang telah diproses agar kelak tidak diproses kembali. Contoh dari algoritma traversal graf adalah algoritma Depth First Search (DFS) dan Breadth-First Search (BFS). Keduanya merupakan algoritma traversal graf populer yang memiliki perbedaan dalam hal urutan pemrosesan vertex.

B. Breadth First Search

Breadth First Search (BFS) atau pencarian melebar merupakan sebuah metode untuk melakukan traversal atau pencarian data pada pohon maupun graf dengan melakukan pencarian pada setiap cabang, jika tidak ditemukan data yang dicari, maka pencarian dilanjutkan dengan traversal kembali setiap cabang yang ada. Berikut ini adalah algoritma untuk melakukan BFS:

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

Algoritma ini dimanfaatkan untuk mencari file yang diinginkan yaitu dengan menggunakan queue yang konsepnya sesuai dengan algoritma BFS, setiap file dari directory yang berada pada parent directory dimasukkan ke dalam queue dan diproses hingga ditemukan file yang diinginkan.

C. Depth First Search

Depth First Search (DFS), pencarian mendalam, merupakan sebuah algoritma untuk pen-traversalan atau pencarian struktur data pohon maupun graph dengan menggunakan aturan backtracking. Algoritma ini akan mencari seluruh vertex/node dengan bergerak lebih dalam jika memungkinkan atau bergerak mundur (backtrack) pada kondisi tertentu. Pada implementasinya, algoritma DFS dapat memanfaatkan struktur data stack. Langkah-langkah utama dari algoritma DFS terdiri dari:

1. Mengunjungi simpul pertama v .

2. Mengunjungi simpul w yang bertetangga dengan simpul v.
3. Mengulangi 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.

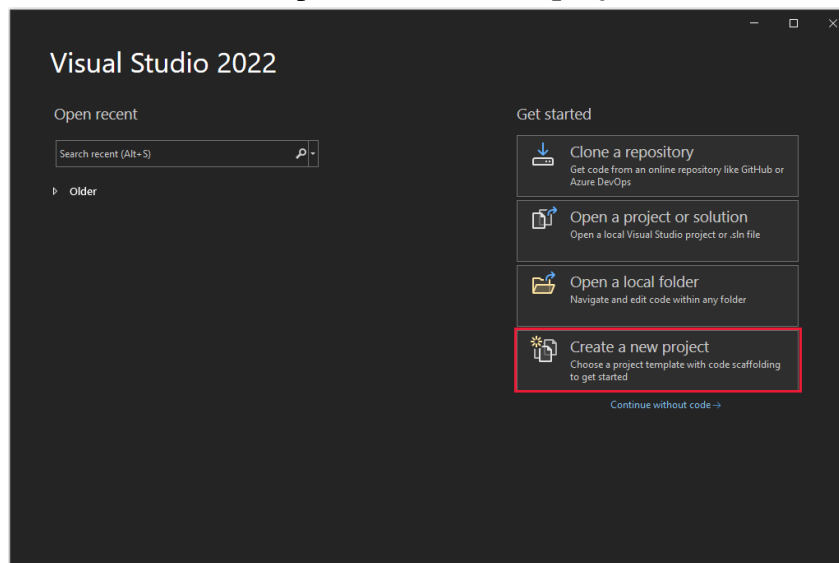
Pada implementasi DFS dengan menggunakan stack, kita dapat menerapkan langkah sebagai berikut:

1. Pilih sebuah simpul awal dan “push” simpul-simpul yang bertetangga dengan simpul awal ke dalam stack.
2. “pop” sebuah simpul dari stack dan “push” semua simpul yang bertetangga dengannya ke dalam stack.
3. Ulangi proses ini sampai stack kosong atau sampai tujuan ditemukan.

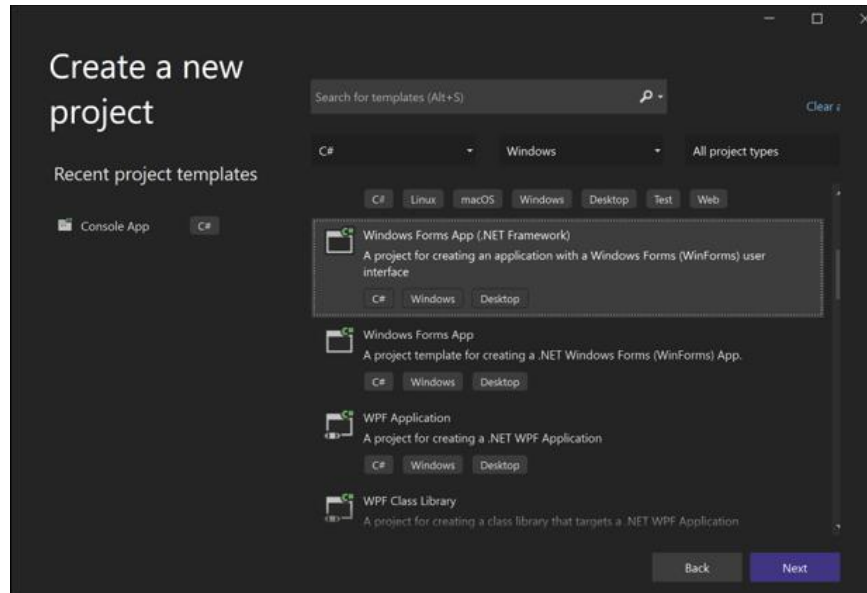
D. C# Desktop Application Development

Dalam mengembangkan aplikasi desktop menggunakan bahasa C#, kita membutuhkan IDE Visual Studio untuk memudahkan pengembangan. Berikut langkah-langkah untuk membuat project windows form pada visual studio:

1. Buka Visual Studio.
2. Pada antarmuka awal, pilih **Create a new project**.

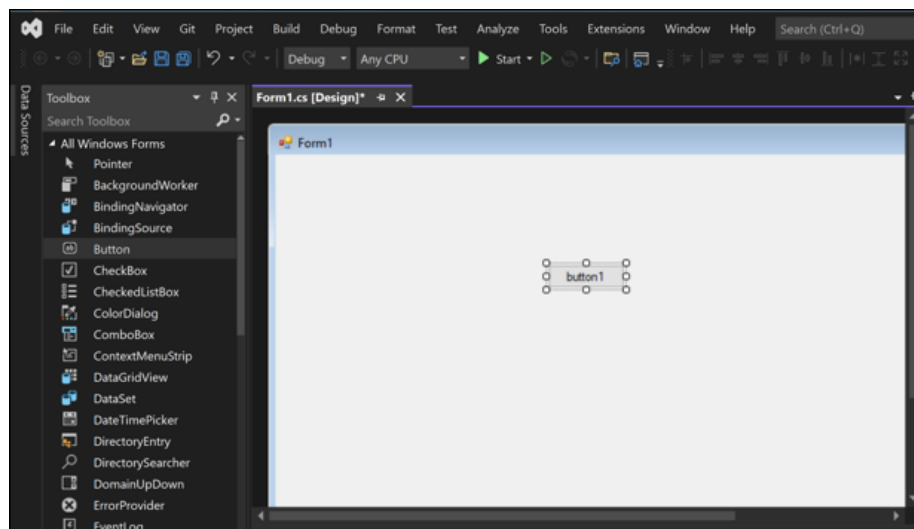


3. Pada jendela **Create a new project**, pilih **Windows Form App (.NET Framework)**

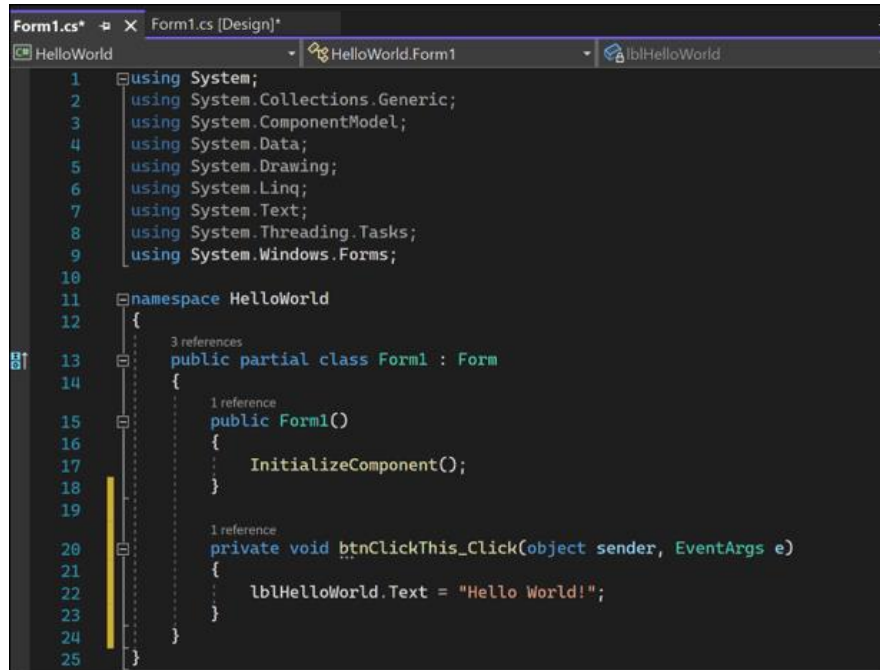


4. Pilih nama project dan versi .NET Framework pada jendela yang muncul.

Pada design form yang diberikan, kita bisa menambah *component/control* seperti text, label, tombol, dan lain-lain pada *toolbar* pada sisi kiri aplikasi visual studio. Kita juga dapat mengedit propertinya seperti posisi, text, ukuran, dan lain-lain.

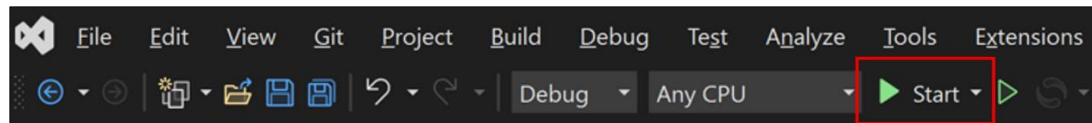


Untuk memprogram aksi yang bisa dilakukan tiap elemen pada form, kita bisa melakukan *coding* pada file *Form.cs*. Contohnya pada “button” kita bisa mengedit aksi yang dilakukan ketika tombol ditekan.



```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Threading.Tasks;
9 using System.Windows.Forms;
10
11 namespace HelloWorld
12 {
13     public partial class Form1 : Form
14     {
15         public Form1()
16         {
17             InitializeComponent();
18         }
19
20         private void btnClickThis_Click(object sender, EventArgs e)
21         {
22             lblHelloWorld.Text = "Hello World!";
23         }
24     }
25 }
```

Untuk menjalankan aplikasi yang telah dibuat, kita bisa mengklik tombol start, bisa dengan mode debugging untuk mencari kode yang error ketika aplikasi diuji dan bisa juga tanpa mode debugging. Untuk mode debugging, kita juga bisa menggunakan fitur *hot reload* untuk melihat perubahan secara langsung ketika kode diubah tanpa perlu melakukan *compile* ulang.



BAB III

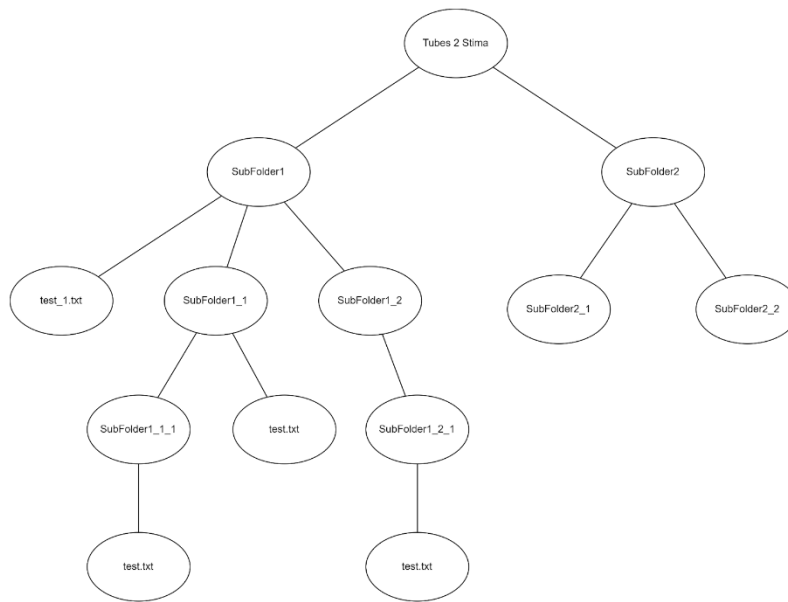
ANALISIS PEMECAHAN MASALAH

Permasalahan yang ingin dipecahkan pada tugas besar ini adalah melakukan pencarian pada filesystem. Filesystem dapat berisi banyak folder dan file yang beragam. Pada dasarnya, sebuah filesystem dapat digambarkan menjadi sebuah pohon n -ary. Hal ini disebabkan oleh karakteristik filesystem yang merupakan rentetan folder dan folder/file yang berada di dalamnya dan folder/file yang berada di dalamnya lagi dan seterusnya.

Oleh sebab itu, pencarian yang akan kita lakukan akan menjadi lebih mudah karena *filesystem* tidak akan membentuk sirkuit sehingga kita tidak perlu khawatir akan terciptanya *loop*. Sebagai contoh, berikut adalah contoh filesystem pada sebuah drive:

```
D:\Tubes 2 Stima>tree /F
Folder PATH listing for volume Dims
Volume serial number is 72CA-0061
D:.\
|_ SubFolder1
|   |_ test_1.txt
|   |_ SubFolder1_1
|       |_ test.txt
|       |_ SubFolder1_1_1
|           |_ test.txt
|   |_ SubFolder1_2
|       |_ SubFolder1_2_1
|           |_ test.txt
|_ SubFolder2
|   |_ SubFolder2_1
|   |_ SubFolder2_2
```

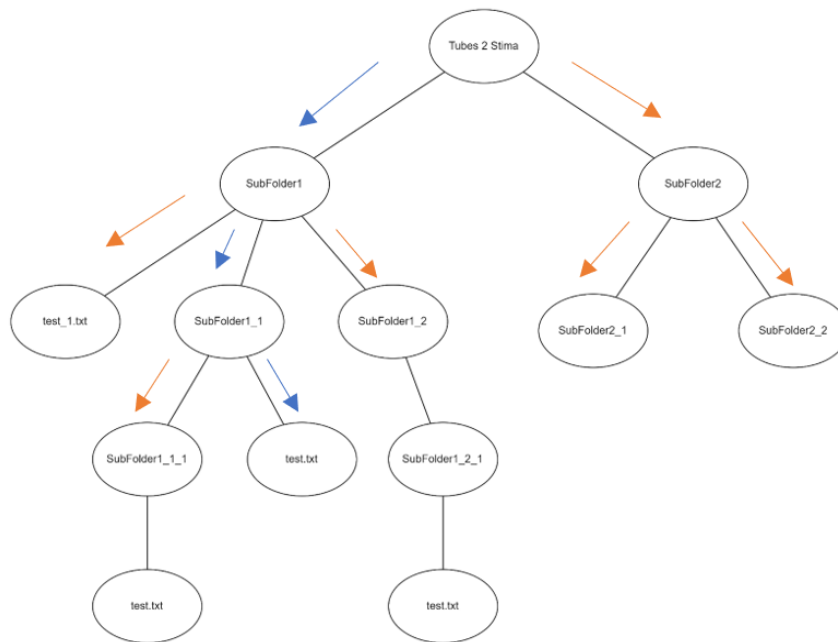
Dapat dilihat bahwa folder “Tubes 2 Stima” menjadi akar dari *filesystem*. *Filesystem* dapat dipetakan menjadi sebuah pohon seperti pada gambar di bawah:



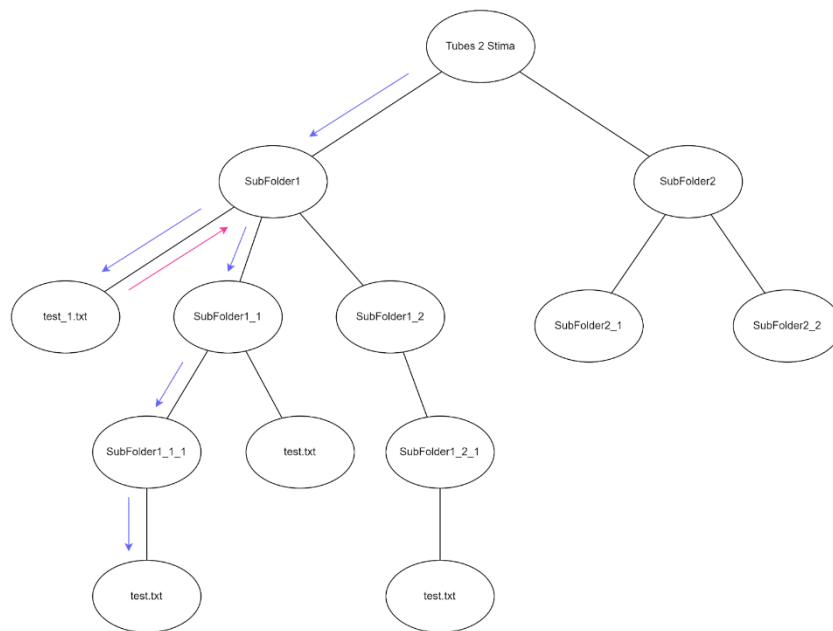
Dari gambar di atas, kita dapat menyimpulkan bahwa *path* dari sebuah file/folder yang ingin dicari pada sebuah *filesystem* merupakan *path* akar diconcat dengan nama-nama simpul yang menjadi lintasan dari akar sampai ke simpul file/folder yang ingin dicari. Dengan menggunakan contoh di atas, *path* dari file “test_1.txt” adalah “D:\Tubes 2 Stima” (*path akar*) diconcat dengan “\SubFolder1” dan “\test_1.txt” menjadi “D:\Tubes 2 Stima\SubFolder1\test_1.txt”.

Pencarian sebuah file dapat dilakukan dengan menggunakan algoritma BFS maupun DFS.

Pada algoritma BFS, file dalam sebuah folder akan ditraversal terlebih dahulu dan dilanjutkan ke setiap folder pada directory yang dipilih. Berikut ini adalah contoh pencarian sebuah file dengan nama “test.txt”



Pada algoritma DFS, simpul akar akan menjadi simpul pertama yang diperiksa. Selanjutnya, pencarian akan dilakukan sesuai langkah pada landasan teori. Berikut adalah contoh pencarian sebuah file bernama “test.txt”.



BAB IV

IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Program

Program Utama

```
String rootFolder
String fileName
Bool findAllOccurrence
DirCrawlerBFS dirCrawlerBFS // Object
DirCrawlerDFS dirCrawlerDFS // Object
String method

procedure folderSelect_Click() {
    // Prosedur untuk memilih rootFolder
    input(rootFolder)
}

procedure placelabel() { /* Prosedur untuk menempatkan hyperlink solusi pada tampilan
aplikasi */}

procedure load_result(String method) { /* Prosedur untuk menyimpan hasil crawling ke
dalam variabel lokal untuk dijadikan hyperlink dan memanggil placelabels()*/}

procedure explore(String method, bool findAll) {
    if (method == "BFS") {
        if (findAll) {
            dirCrawlerBFS.searchAll(rootFolder, fileName)
        } else {
            dirCrawlerBFS.searchFirst(rootFolder, fileName)
        }
    } else if (method == "DFS") {
        if (findAll) {
            dirCrawlerDFS.searchAll(rootFolder, fileName)
        } else {
            dirCrawlerDFS.searchFirst(rootFolder, fileName)
        }
    }
    Load_result(method)
}
```



```
procedure searchButton_Click() {  
    /* Prosedur ketika tombol search ditekan */  
    findAllOccurence = this.findAllCheckbox.Checked  
    if (this.bfsRadioButton.Checked) // Method bfs dipilih pada UI  
    {
```

}

}

```
        dirCrawlerBFS = new DirCrawlerBFS() // Inisiasi object  
        explore("BFS", findAllOccurence)  
        graphTree = dirCrawlerBFS.Graph // Menyimpan graf  
    }  
    else  
    {  
        dirCrawlerDFS = new DirCrawlerDFS() // Inisiasi object  
        explore("DFS", findAllOccurence)  
        graphTree = dirCrawlerDFS.Graph // Menyimpan graf  
    }  
}
```

}

Algoritma Breadth First Search

SearchFirst:

```
procedure searchFirst(input rootPath: string, fileTarget: string)
{ Traversal dengan metode BFS
    Masukan: Directory searching dimulai dan file yang dicari
    Keluaran: Pohon berwarna yang menunjukkan jalur pencarian serta hyperlink directory
    ditemukannya file
}
Deklarasi
    q : antrian;
    results : array of string; {array untuk menyimpan path ditemukannya file}
    currDirs : array of string; {array untuk menyimpan dirs yang ada di dirs saat ini}
    currFiles : array of string; {array untuk menyimpan file yang ada di dirs saat ini}
    found : boolean
    procedure BuatAntrian(input/output q: antrian)
    {membuat antrian kosong}
    procedure MasukAntrian (input/output q: antrian, input v: string)
    {memasukkan v ke dalam antrian q pada posisi belakang}
    procedure HapusAntrian (input/output q: antrian, output v: string)
    {menghapus v dari kepala antrian q}
    function AntrianKosong (input q:antrian) → boolean
    { true jika antrian q kosong, false jika sebaliknya}
    function GetDirectories (input currPath: string) → array of string
    {mengembalikan daftar directories yang terdapat dalam masukan currPath}
    function GetFiles (input currPath: string) → array of string
    {mengembalikan daftar files yang terdapat dalam masukan currPath}
    procedure drawPohon(input currPath: string, input file: string)
    {memasukkan sisi jika belum ada ke dalam pohon}
    procedure giveColor (input currPath: string, input rootPath: string, input color: string)
    {memberi pada sisi yang dilalui currPath hingga rootPath dengan warna color}
```

Algoritma

```
BuatAntrian(q)
MasukAntrian(q, rootPath)
found ← false
while (not AntrianKosong(q) and not found) do
    currPath = HapusAntrian(q, v)
    currDirs = GetDirectories(currPath)
    currFiles = GetFiles(currPath)
    for file in currFiles do
        MasukAntrian(q, file)
        drawPohon(currPath, file)
    for dir in currDirs do
        MasukAntrian(q, dir)
        drawPohon(currPath, dir)
    if (fileTarget = GetFileName(currPath)) do
        results ← results + currPath
        found ← true
    else
        giveColor(currPath, rootPath, "red")
for result in results do
    giveColor(result, rootPath, "blue") {memberi warna biru pada rute hasil}
```

SearchAll:

Tugas Besar 2 IF2211 Strategi Algoritma
Kelompok Dora the Explowew

```
procedure searchFirst(input rootPath: string, fileTarget: string)
{ Traversal dengan metode BFS
  Masukan: Directory searching dimulai dan file yang dicari
  Keluaran: Pohon berwarna yang menunjukkan jalur pencarian serta hyperlink directory
             ditemukannya file
}
Deklarasi
  q : antrian;
  results : array of string; {array untuk menyimpan path ditemukannya file}
  currDirs : array of string; {array untuk menyimpan dirs yang ada di dirs saat ini}
  currFiles : array of string; {array untuk menyimpan file yang ada di dirs saat ini}
  procedure BuatAntrian(input/output q: antrian)
  {membuat antrian kosong}
  procedure MasukAntrian (input/output q: antrian, input v: string)
  {memasukkan v ke dalam antrian q pada posisi belakang}
  procedure HapusAntrian (input/output q: antrian, output v: string)
  {menghapus v dari kepala antrian q}
  function AntrianKosong (input q:antrian) → boolean
  { true jika antrian q kosong, false jika sebaliknya}
  function GetDirectories (input currPath: string) → array of string
  {mengembalikan daftar directories yang terdapat dalam masukan currPath}
  function GetFiles (input currPath: string) → array of string
  {mengembalikan daftar files yang terdapat dalam masukan currPath}
  procedure drawPohon(input currPath: string, input file: string)
  {memasukkan sisi jika belum ada ke dalam pohon}
  procedure giveColor (input currPath: string, input rootPath: string, input color: string)
  {memberi pada sisi yang dilalui currPath hingga rootPath dengan warna color}
Algoritma
```

```
BuatAntrian(q)
MasukAntrian(q, rootPath)
while (not AntrianKosong(q)) do
    currPath = HapusAntrian(q, v)
    currDirs = GetDirectories(currPath)
    currFiles = GetFiles(currPath)
    for file in currFiles do
        MasukAntrian(q, file)
        drawPohon(currPath, file)
    for dir in currDirs do
        MasukAntrian(q, dir)
        drawPohon(currPath, dir)
    if (fileTarget = GetFileName(currPath)) do
        results ← results + currPath
    else
        giveColor(currPath, rootPath, "red")
for result in results do
    giveColor(result, rootPath, "blue") {memberi warna biru pada rute hasil}
```

Algoritma Depth First Search:

1. Search First:

```

procedure searchFirst(input rootPath: string, fileTarget: string)
{ Traversal dengan metode DFS
    Masukan: Directory searching dimulai dan file yang dicari
    Keluaran: Pohon berwarna yang menunjukkan jalur pencarian serta hyperlink directory
    ditemukannya file.
}

Deklarasi
s: stack;
results: array of string; {array untuk menyimpan path ditemukannya file}
currDirs: array of string; {array untuk menyimpan dirs yang ada di dirs saat ini}
currFiles: array of string; {array untuk menyimpan file yang ada di dirs saat ini}
found: boolean
procedure BuatStack(input/output s: stack)
{membuat stack kosong}
procedure push (input/output s: stack, input v: string)
{memasukkan v ke dalam stack s pada posisi depan}
function pop (input/output s: stack)
{mengeluarkan dari top dari stack}
function StackKosong (input s: stack) → boolean
{ true jika stack s kosong, false jika sebaliknya}
function GetDirectories (input currPath: string) → array of string
{mengembalikan daftar directories yang terdapat dalam masukan currPath}
function GetFiles (input currPath: string) → array of string
{mengembalikan daftar files yang terdapat dalam masukan currPath}
procedure drawPohon(input currPath: string, input file: string)
{memasukkan sisi jika belum ada ke dalam pohon}
procedure giveColor (input currPath: string, input rootPath: string, input color: string)
{memberi pada sisi yang dilalui currPath hingga rootPath dengan warna color}

```

Algoritma

```
BuatStack(s)
push(s, rootPath)
found ← false
while (not StackKosong(s) and not found) do
    currPath = pop(s)
    currDirs = GetDirectories(currPath)
    currFiles = GetFiles(currPath)
    for file in currFiles do
        push(s, file)
        drawPohon(currPath, file)
    for dir in currDirs do
        push(s, dir)
        drawPohon(currPath, dir)
    if (fileTarget = GetFileName(currPath)) do
        results ← results + currPath
        found ← true
    else
        giveColor(currPath, rootPath, "red")
for result in results do
    giveColor(result, rootPath, "blue") {memberi warna biru pada rute hasil}
```

2. Search All:

SEARCHALL

```
procedure searchFirst(input rootPath: string, fileTarget: string)
{ Traversal dengan metode DFS
    Masukan: Directory searching dimulai dan file yang dicari
    Keluaran: Pohon berwarna yang menunjukkan jalur pencarian serta hyperlink directory
    ditemukannya file
}
```

Deklarasi

```
s: stack;
results: array of string; {array untuk menyimpan path ditemukannya file}
currDirs: array of string; {array untuk menyimpan dirs yang ada di dirs saat ini}
currFiles: array of string; {array untuk menyimpan file yang ada di dirs saat ini}
procedure BuatStack(input/output s: stack)
{membuat stack kosong}
procedure push (input/output s: stack, input v: string)
{memasukkan v ke dalam stack s pada posisi depan}
function pop (input/output q: stack)
{mengembalikan top dari stack s}
function StackKosong (input s: stack) → boolean
{ true jika stack s kosong, false jika sebaliknya}
function GetDirectories (input currPath: string) → array of string
{mengembalikan daftar directories yang terdapat dalam masukan currPath}
function GetFiles (input currPath: string) → array of string
{mengembalikan daftar files yang terdapat dalam masukan currPath}
procedure drawPohon(input currPath: string, input file: string)
{memasukkan sisi jika belum ada ke dalam pohon}
procedure giveColor (input currPath: string, input rootPath: string, input color: string)
{memberi pada sisi yang dilalui currPath hingga rootPath dengan warna color}
```


Algoritma

```
BuatStack(s)
push(s, rootPath)
while (not StackKosong(s)) do
    currPath = pop(s)
    currDirs = GetDirectories(currPath)
    currFiles = GetFiles(currPath)
    for file in currFiles do
        push(s, file)
        drawPohon(currPath, file)
    for dir in currDirs do
        push(s, dir)
        drawPohon(currPath, dir)
    if (fileTarget = GetFileName(currPath)) do
        results ← results + currPath
    else
        giveColor(currPath, rootPath, "red")
    for result in results do
        giveColor(result, rootPath, "blue") {memberi warna biru pada rute hasil}
```

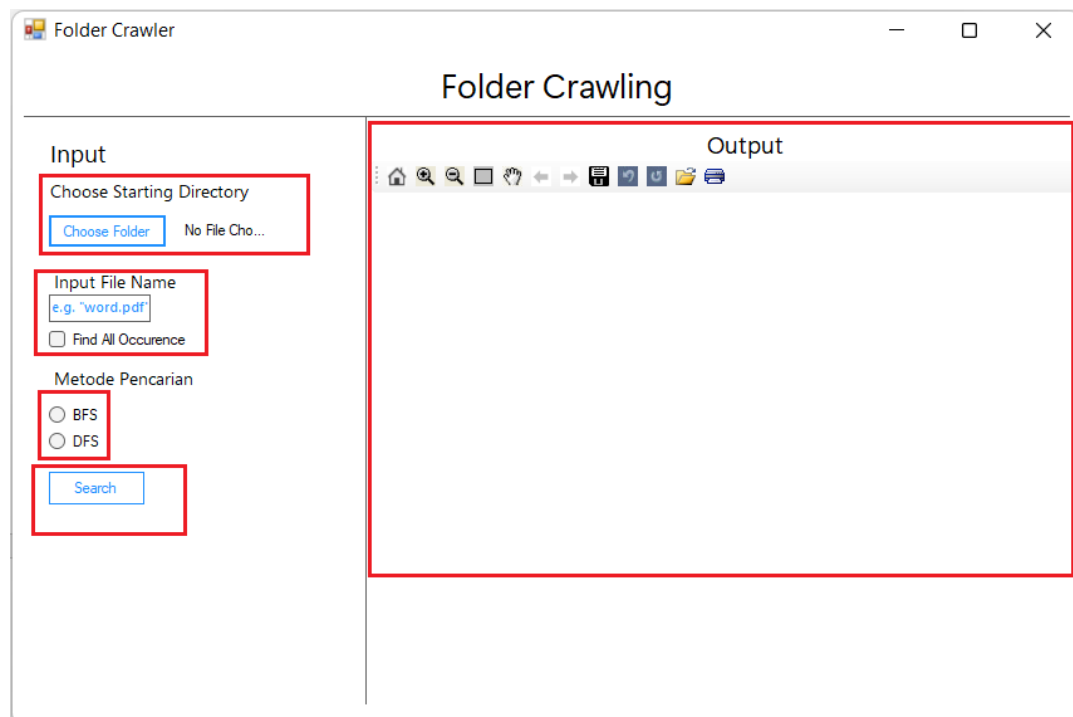
B. Penjelasan Struktur Data

1. DirCrawlerBFS.cs
File ini berisi kelas DirCrawlerBFS yang berisi implementasi dari algoritma Breadth First Search dalam bahasa C#.
2. DirCrawlerDFS.cs
File ini berisi kelas DirCrawlerDFS yang berisi implementasi dari algoritma Depth First Search dalam bahasa C#.
3. Program.cs
File ini berisi kelas Program yang merupakan entry point utama dari aplikasi.
4. UI.Designer.cs
File ini berisi kode yang digenerate dari komponen-komponen beserta propertinya yang telah diinisiasi ke dalam program. Kita juga bisa mengedit atribut tiap komponen secara manual dari file ini.
5. UI.cs

File ini berisi kelas UI yang merupakan alur utama dari program yang telah dibuat mulai dari input folder *root*, nama file, metode pencarian, dan aksi ketika tombol search ditekan.

C. Cara Penggunaan Program

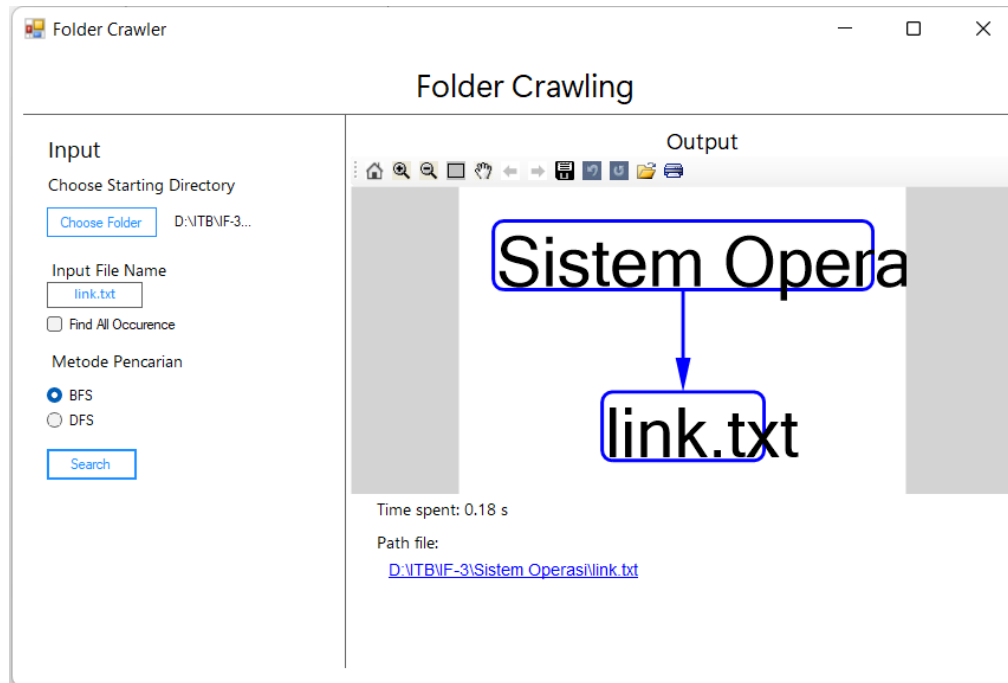
Berikut adalah User Interface program ketika pertama kali dijalankan.



Komponen-komponen yang terdapat pada UI program antara lain:

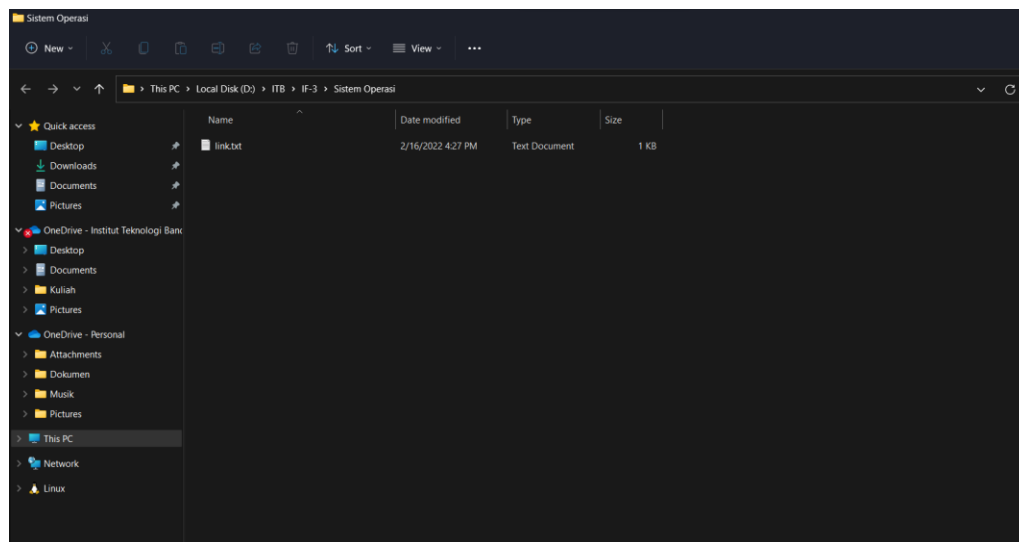
1. Tombol **Choose Folder**, yang berfungsi untuk memilih root folder pencarian file.
2. Input Textbox, yaitu tempat untuk menuliskan nama file yang ingin kita cari.
3. Checkbox **Find All Occurrence**, untuk menandai apakah kita ingin mencari semua file dengan nama tertentu atau hanya hasil pencarian pertama saja.
4. Radio button **Metode Pencarian**, untuk menentukan pilihan metode pencarian yang digunakan (BFS atau DFS).
5. Button **Search**, untuk memulai pencarian file dari suatu root folder dengan metode yang telah dipilih.
6. Gviewer output, untuk menampilkan pohon pencarian yang telah dibangun setelah selesai melakukan pencarian file. Ketika pencarian berhasil, maka akan ditampilkan juga *hyperlink* ke folder parent file yang sedang dicari.

Berikut adalah contoh program yang telah dijalankan.



Graph viewer yang digunakan juga memiliki fitur untuk melakukan *zoom in*, *zoom out*, *geser*, *save*, dan lain-lain untuk memudahkan user dalam mengolah hasil keluaran.

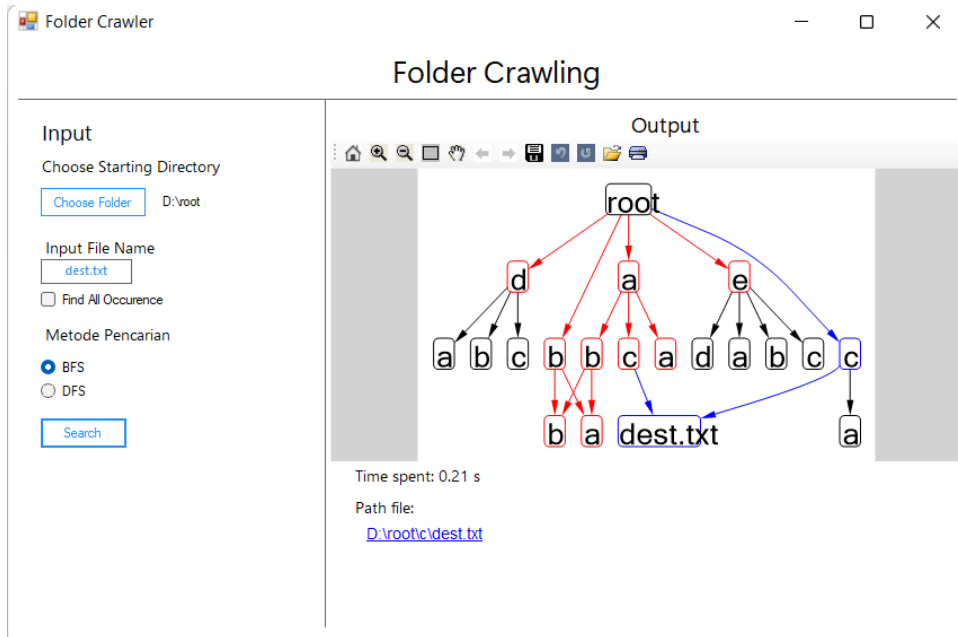
Jika user mengklik hyperlink yang ditampilkan, maka system akan membuka explorer dan menuju folder parent dari file yang dicari.



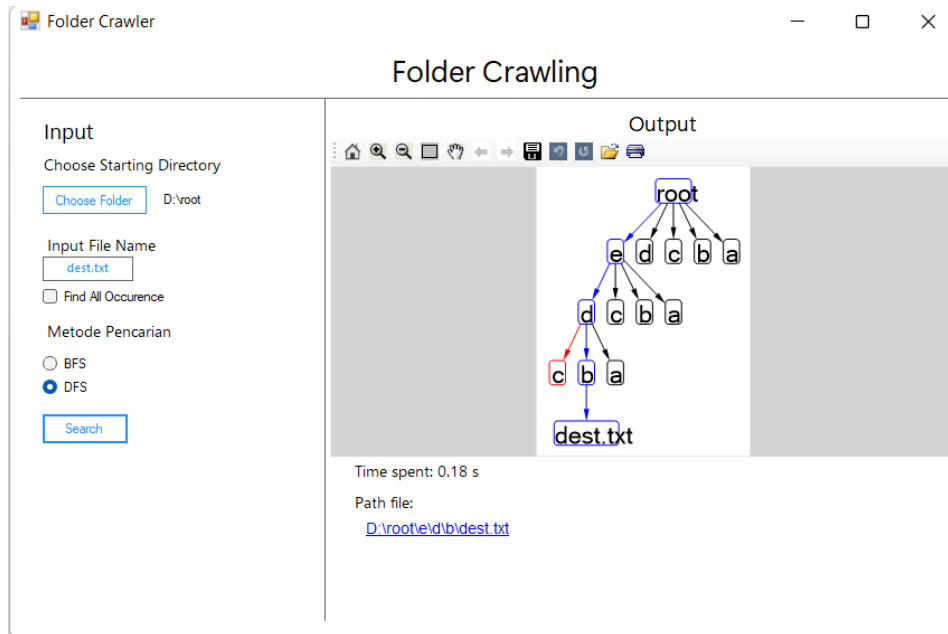
D. Hasil Pengujian

Berikut adalah hasil pengujian yang dilakukan pada folder root dengan nama file yang dicari adalah dest.txt:

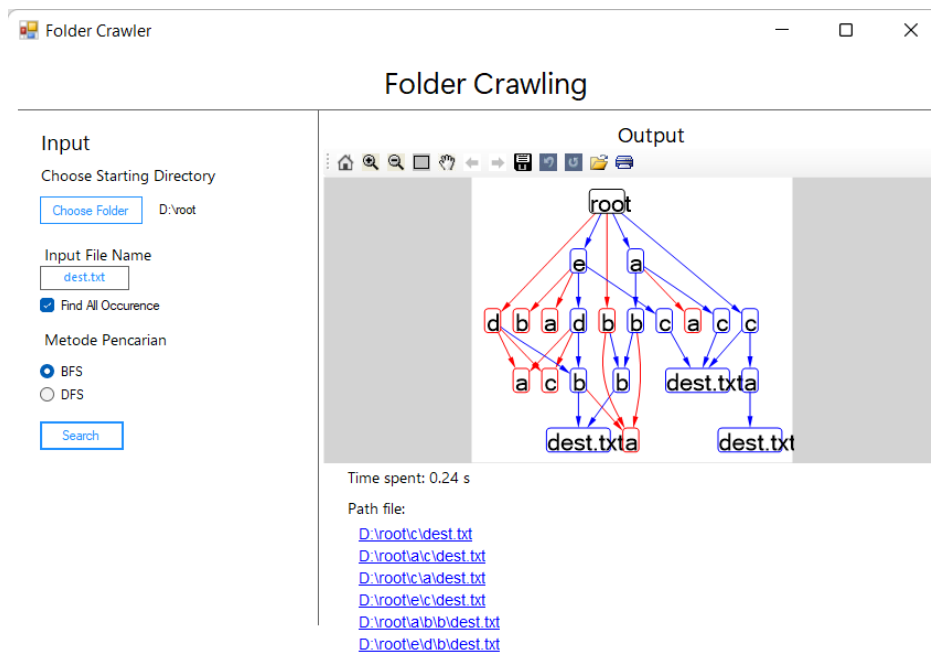
Folder	D:\root
Nama File	dest.txt
Find All Occurrence	false
Metode	BFS



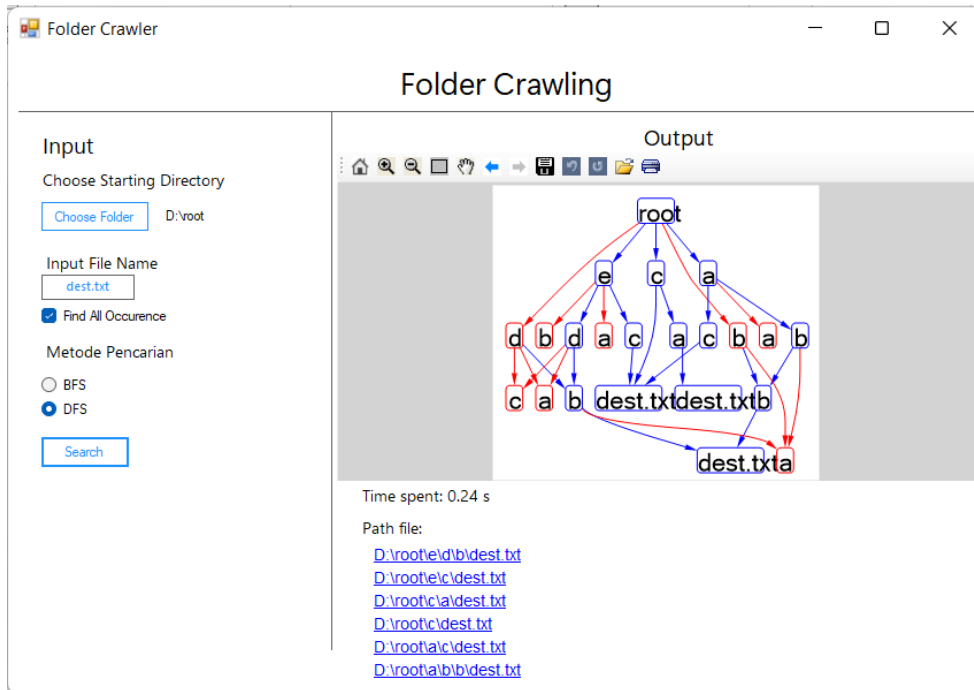
Folder	D:\root
Nama File	dest.txt
Find All Occurrence	false
Metode	DFS



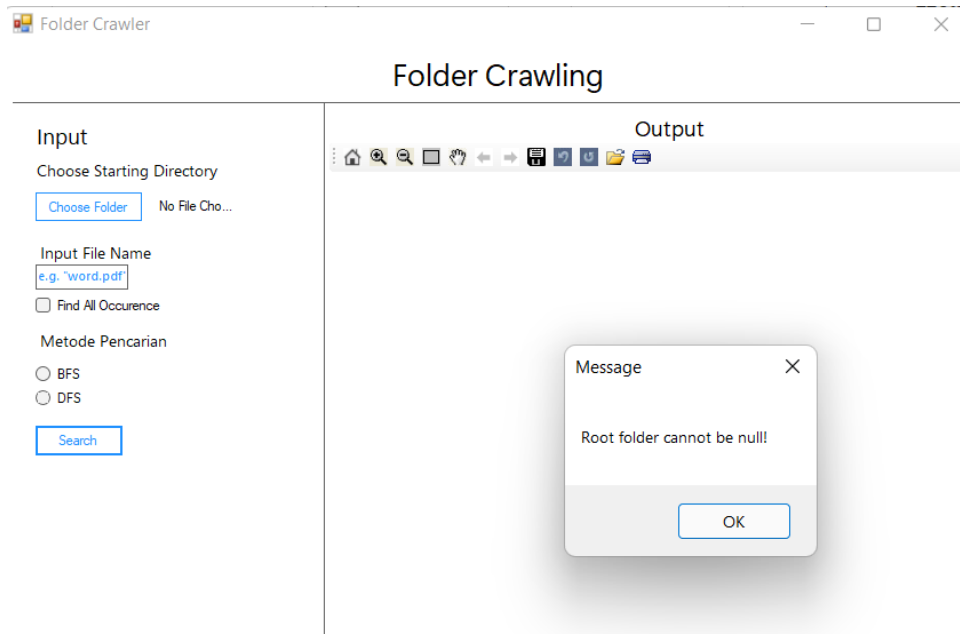
Folder	D:\root
Nama File	dest.txt
Find All Occurrence	true
Metode	BFS



Folder	D:\root
Nama File	dest.txt
Find All Occurrence	true
Metode	DFS



Program juga dapat memberikan pesan kesalahan ketika input yang diberikan tidak lengkap seperti pada gambar di bawah ini.



E. Analisis Desain Solusi Algoritma BFS dan DFS

Dari hasil uji coba yang dilakukan, program sudah dapat melakukan folder crawling dalam pencarian file menggunakan algoritma BFS maupun DFS. Jika dibandingkan efisiensi dari kedua algoritma tersebut, pada pencarian file pertama, BFS lebih cepat melakukan pencarian solusi yang letaknya tidak jauh dari folder parent, meskipun jumlah folder child didalamnya sangat banyak karena algoritma BFS yang akan melakukan pencarian pada semua folder pada kedalaman yang sama terlebih dahulu. Sedangkan jika jumlah folder child tidak terlalu banyak tetapi letaknya cukup dalam, algoritma DFS bekerja lebih efisien karena algoritma ini akan mencari solusi sampai kedalaman maksimum lebih dahulu daripada memeriksa folder lain dengan kedalaman yang sama.

Untuk pencarian semua file yang memiliki nama yang cocok, kedua algoritma mempunyai performa yang cukup seimbang dengan asumsi file yang dicari tersebar merata dari kedalamannya karena kedua algoritma akan sama-sama menelusuri setiap file dan folder yang terletak di dalam folder parent.

Untuk kasus yang khusus, jika child pada suatu folder sangat banyak, algoritma BFS mungkin saja bekerja secara kurang optimal karena membutuhkan memory yang sangat besar untuk menyimpan antrian yang ada.

BAB V

KESIMPULAN DAN SARAN

A. Kesimpulan

Dari pengerjaan Tugas Besar 2 Strategi Algoritma, kami telah berhasil membuat sebuah aplikasi desktop dalam bahasa C# dengan menggunakan .NET framework yang mengimplementasikan algoritma BFS dan DFS dalam folder crawling. Dengan memasukkan folder utama dan nama file yang ingin dicari, program dapat mencari jalur (*path*) menuju file tujuan yang dimaksud dengan menggunakan salah satu algoritma BFS atau DFS.

B. Saran

1. Graph viewer dapat diatur lebih jauh agar hasil tampilan pohon lebih jelas ketika menampilkan pohon dengan ukuran yang sangat besar.
2. Posisi setiap komponen pada User Interface (UI) dapat dibuat lebih rapi.
3. Pewarnaan pada graph dapat dibuat lebih konsisten agar memudahkan melihat jalur solusi yang benar.

DAFTAR PUSTAKA

1. GRAPHS: basics, representation, traversals, and applications
<https://www.educative.io/edpresso/graphs-basics-representation-traversals-and-applications>
(diakses pada tanggal 15 Maret 2022)
2. Create a Windows Forms app in Visual Studio with C# <https://docs.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022> (diakses pada tanggal 19 Maret 2022)

Link github: https://github.com/louisyyyy/Tubes-2_Dora-the-Explowew

Link video : <https://youtu.be/VLci3oAuABI>