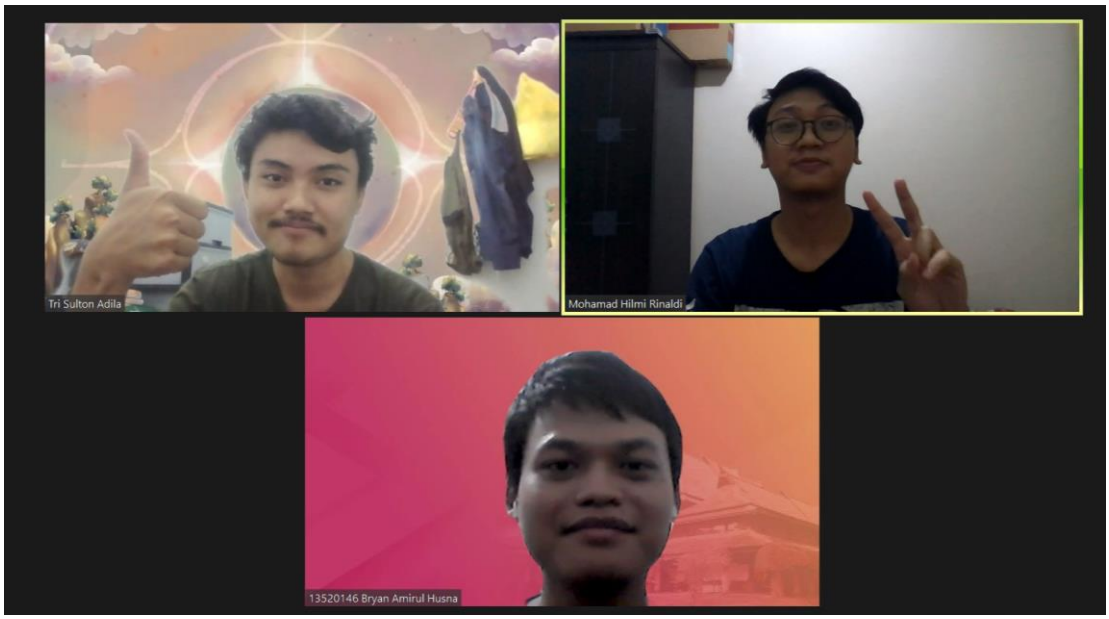
TUGAS BESAR 2 IF 2211

STRATEGI ALGORITMA

Oleh

Tri Sulton Adila	**13520033**
Bryan Amirul Husna	**13520146**
Mohamad Hilmi Rinaldi	**13520149**



PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

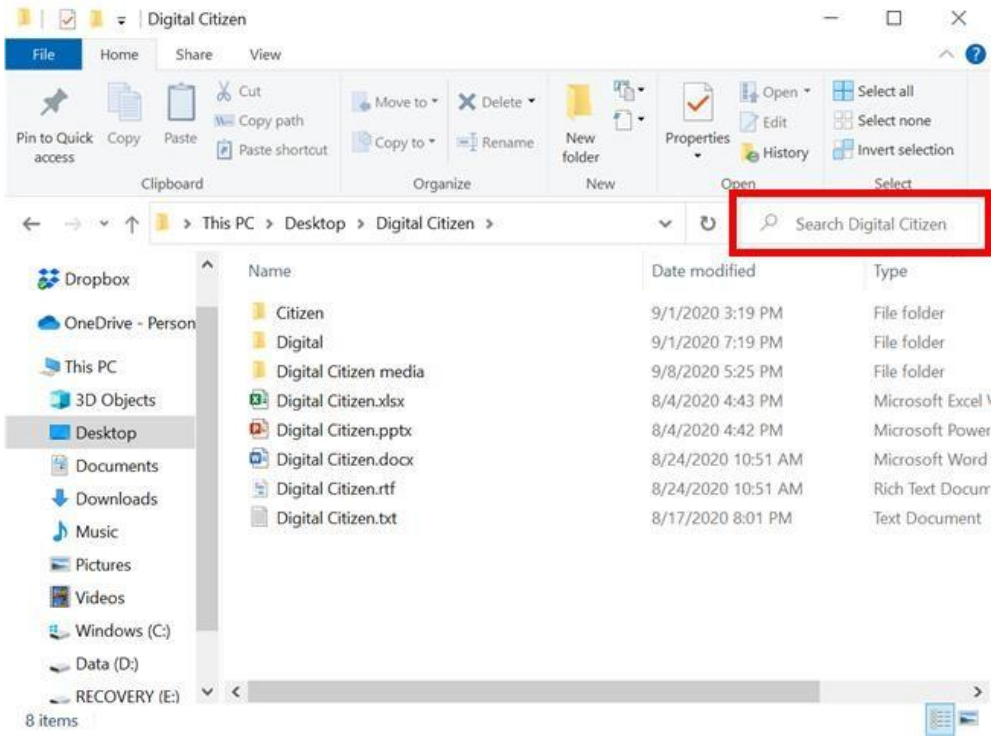
2022

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Latar belakang :

Pada saat kita ingin mencari file spesifik yang tersimpan pada komputer kita, seringkali task tersebut membutuhkan waktu yang lama apabila kita melakukannya secara manual. Bukan sajaharus membuka beberapa folder hingga dapat mencapai directory yang diinginkan, kita bahkan dapat lupa di mana kita meletakkan file tersebut. Sebagai akibatnya, kita harus membuka berbagai folder secara satu persatu hingga kita menemukan file yang diinginkan. Hal ini pastinyaakan sangat memakan waktu dan energi.



Gambar 1. Fitur Search pada Windows 10 File Explorer
(Sumber: https://www.digitalcitizen.life/wp-content/uploads/2020/10/explorer_search_10.png)

Meskipun demikian, kita tidak perlu cemas dalam menghadapi persoalan tersebut sekarang. Pasalnya, hampir seluruh sistem operasi sudah menyediakan fitur *search* yang dapat digunakanuntuk mencari file yang kita inginkan. Kita cukup memasukkan *query* atau kata kunci pada kotakpencarian, dan komputer akan mencarikan seluruh file pada suatu *starting directory* (hingga seluruh *children*-nya) yang berkorespondensi terhadap *query* yang kita masukkan.

Fitur ini diimplementasikan dengan teknik *folder crawling*, di mana mesin komputer akan mulai mencari file yang sesuai dengan *query* mulai dari *starting directory* hingga seluruh *children* dari *starting directory* tersebut sampai satu file pertama/seluruh file ditemukan atau tidak ada file yang ditemukan. Algoritma yang dapat dipilih untuk melakukan *crawling* tersebut pun dapat bermacam-macam dan setiap algoritma akan memiliki teknik dan konsekuensinya sendiri. Oleh karena itu, penting agar komputer memilih algoritma yang tepat sehingga hasil yang diinginkan dapat ditemukan dalam waktu yang singkat.

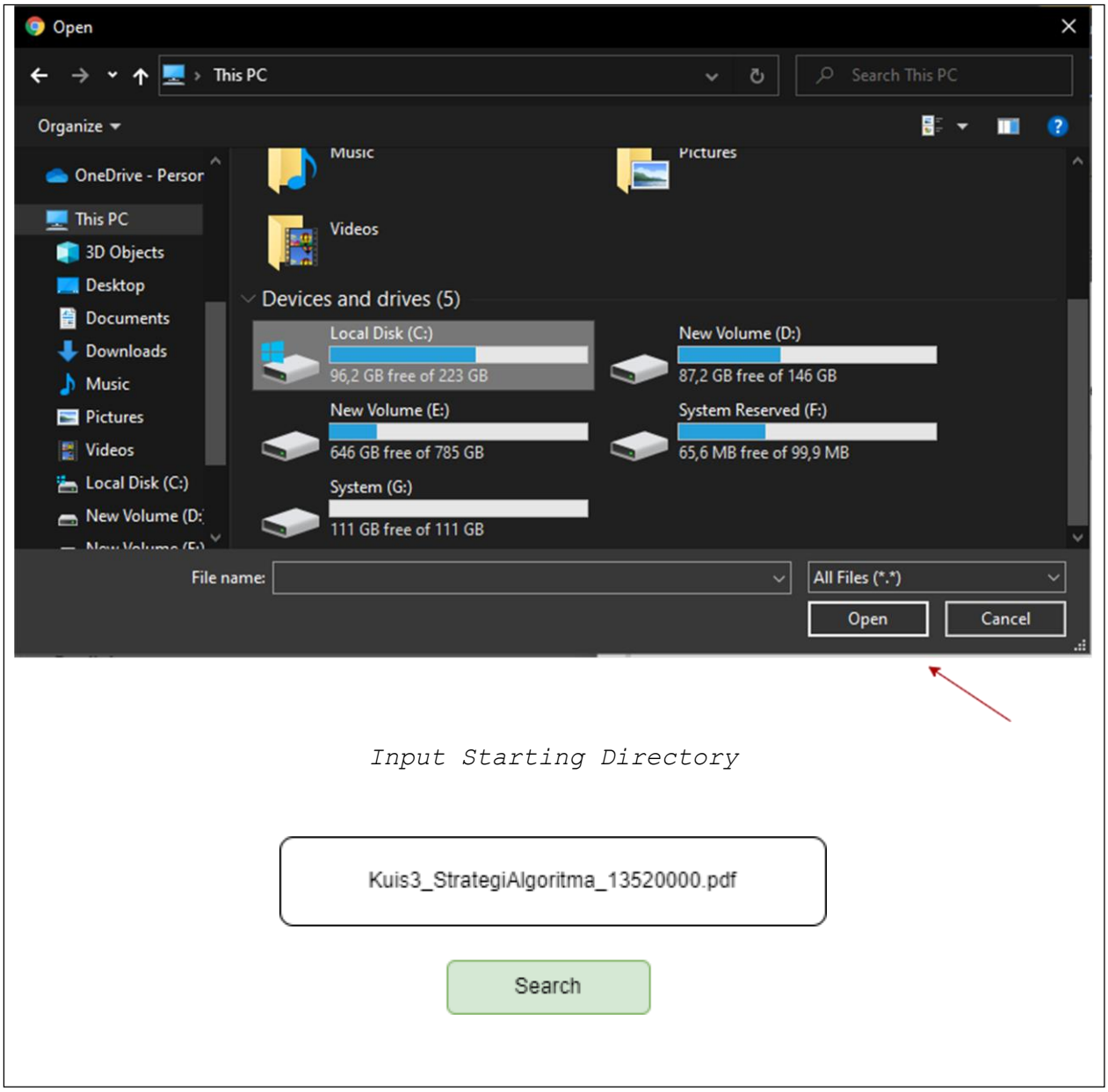
Deskripsi tugas:

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 *DepthFirst Search* (DFS), Anda dapat menelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang Anda inginkan. Anda juga diminta untuk memvisualisasikan hasil daripencarian *folder* tersebut dalam bentuk pohon.

Selain pohon, Anda diminta juga menampilkan list *path* dari daun-daun yang bersesuaian denganhasil 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.

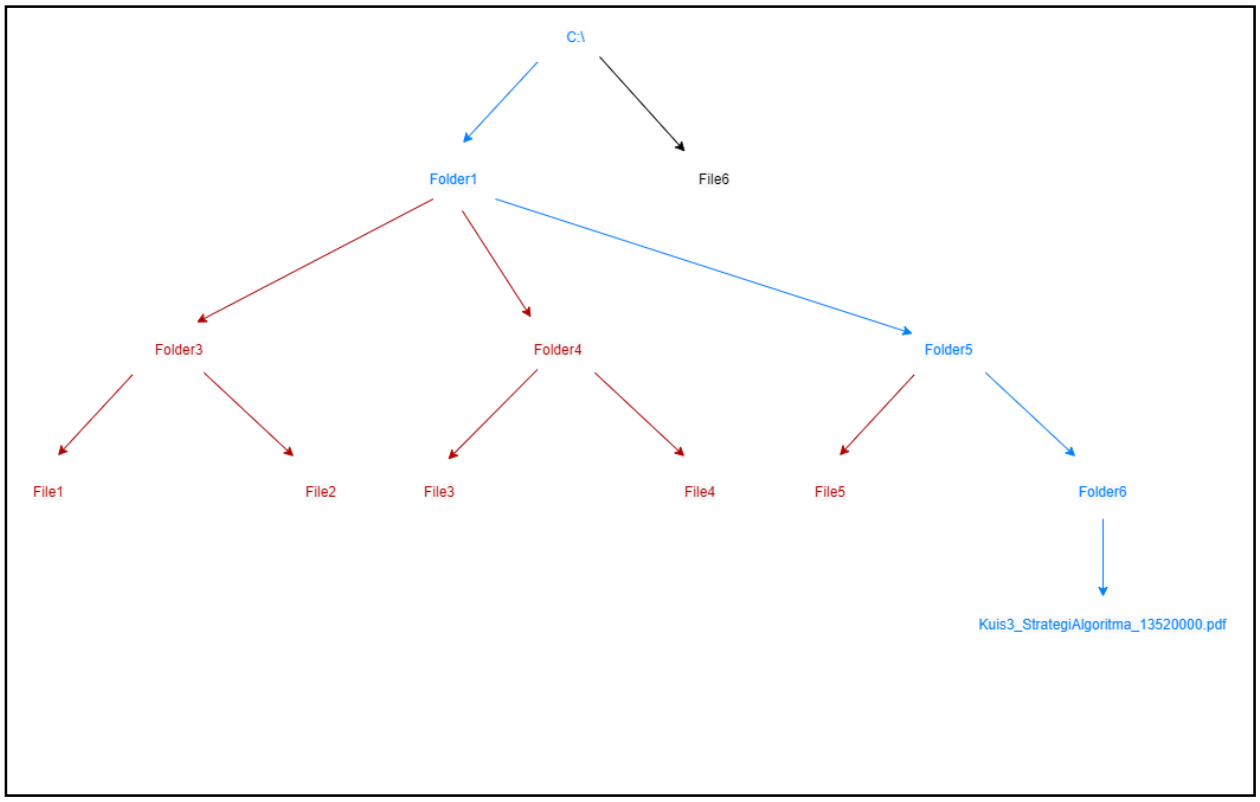
Contoh Input dan Output Program

Contoh masukan aplikasi:



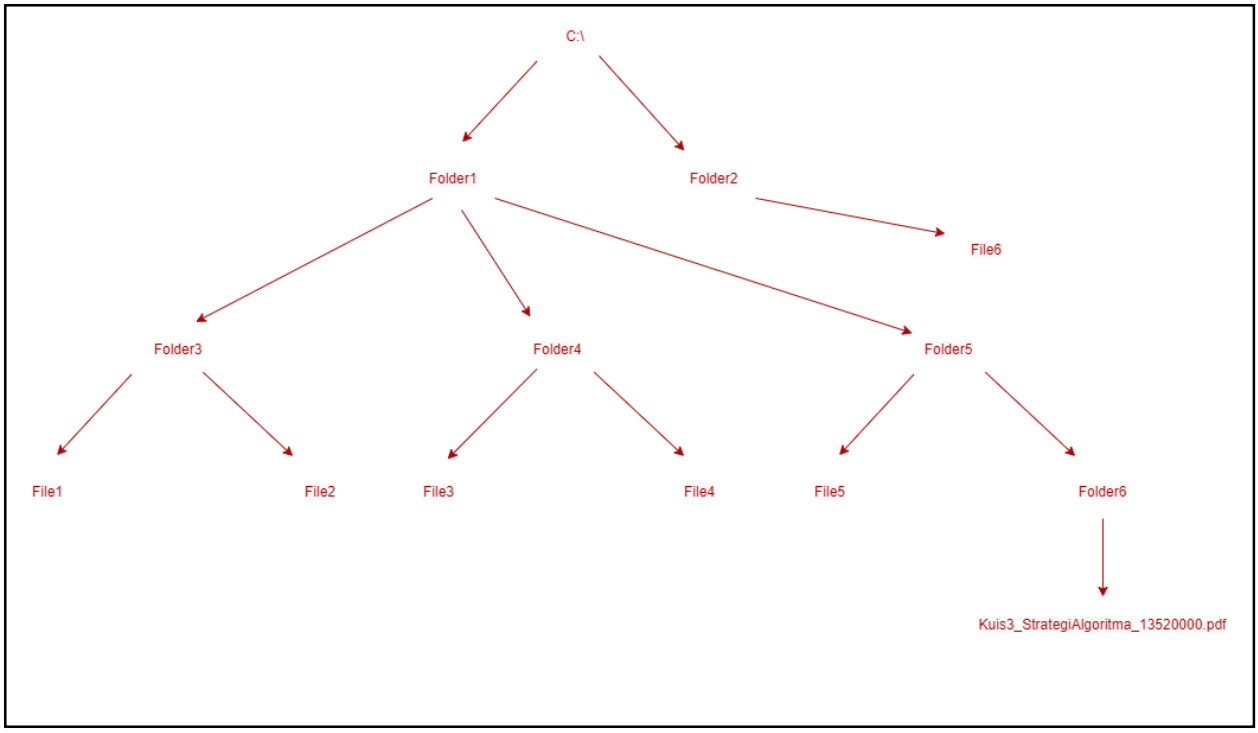
Gambar 2. Contoh input program

Contoh output aplikasi:



Gambar 3. Contoh output program

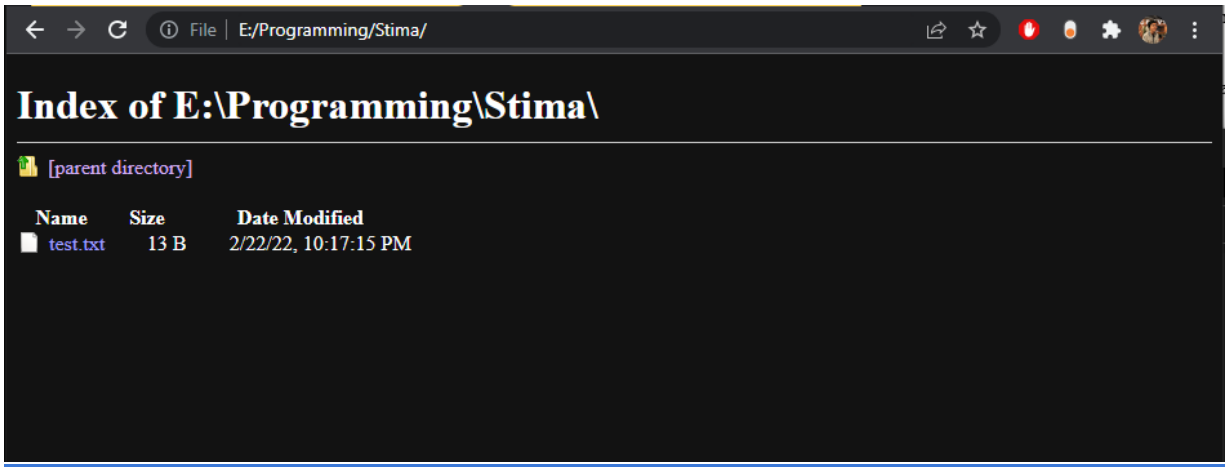
Misalnya pengguna ingin mengetahui langkah *folder crawling* untuk menemukan file Kuis3_StrategiAlgoritma_13520000.pdf.
Maka, path pencarian DFS adalah sebagai berikut. C:\ → Folder1 → Folder3 → File1 → Folder3
→ File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → 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.



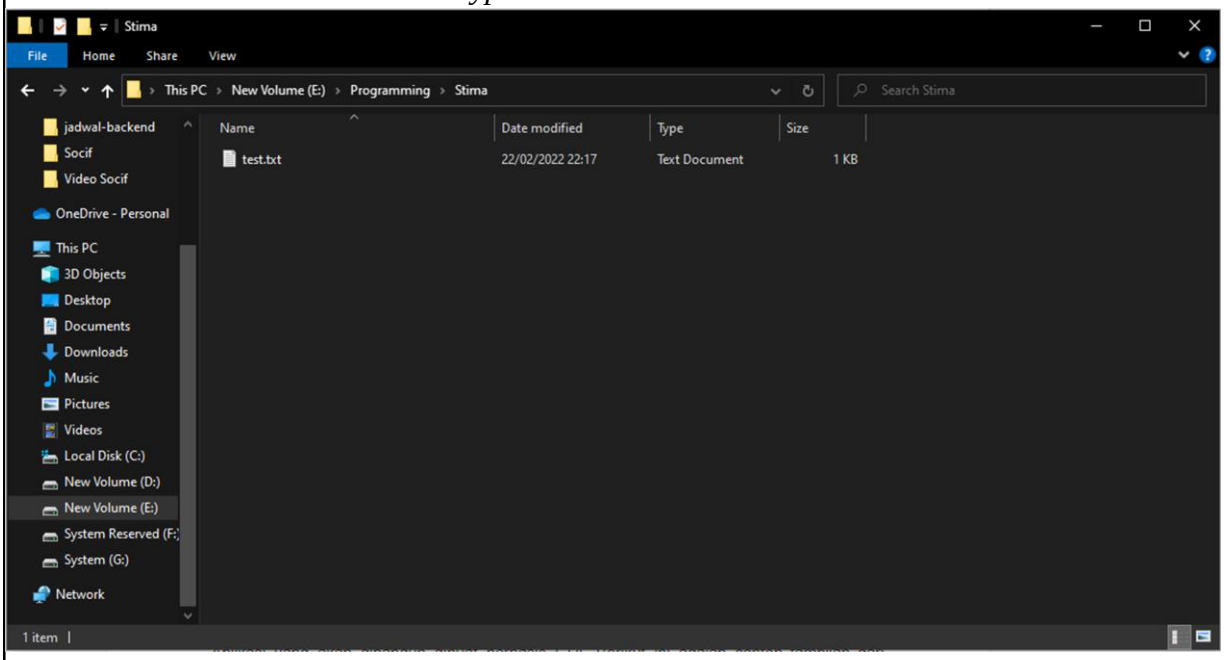
Gambar 4. Contoh output program jika file tidak ditemukan

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:



Contoh Hyperlink Dibuka Melalui Browser



Contoh Hyperlink Dibuka Melalui Folder

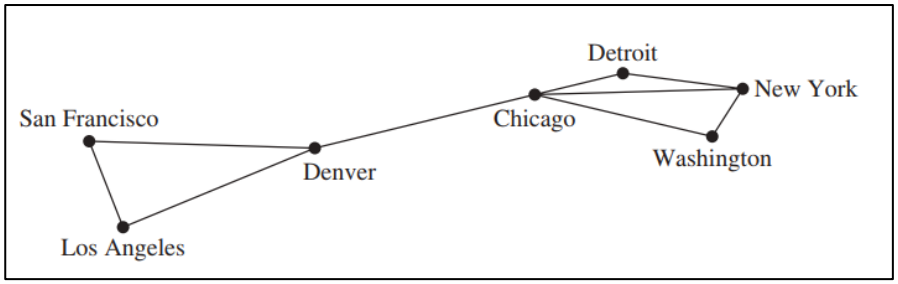
Gambar 5. Contoh ketika hyperlink di-klik

BAB II
LANDASAN TEORI

2.1. Dasar Teori

2.1.1. Graf

Suatu graf didefinisikan sebagai himpunan verteks dan himpunan sisi (edge). Verteks menyatakan entitas-entitas data dan sisi menyatakan keterhubungan antara verteks. Sering kali suatu graf G dinyatakan dalam notasi matematis sebagai G = (V,E) dengan V adalah himpunan verteks dan E adalah himpunan sisi yang terdefinisi antara pasangan-pasangan verteks. Sebuah sisi antara verteks x dan y ditulis sebagai {x, y}.

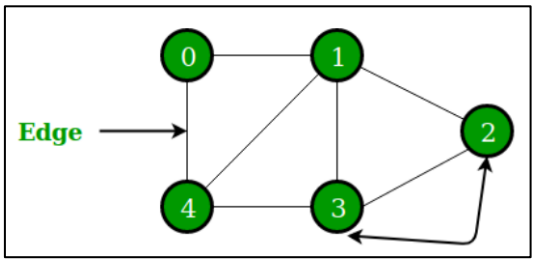


Gambar 6. Representasi Jaringan Komputer sebagai Graf

Graf dapat dikelompokkan menurut keberadaan gelang dan sisi ganda. Gelang merupakan sisi yang menghubungkan suatu verteks dengan dirinya sendiri sehingga terbentuk sebuah loop edge, sedangkan sisi ganda merupakan sisi berjumlah dua atau lebih yang mengikat 2 verteks sekaligus. Berikut pembagian kelompok pada graf.

1. Graf Sederhana

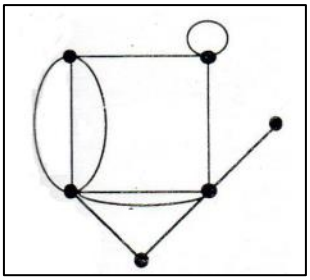
   Graf sederhana tidak mengandung gelang maupun sisi ganda seperti pada gambar berikut.



Gambar 7. Graf Sederhana

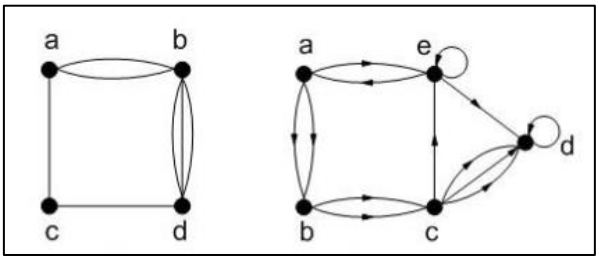
2. Graf Tak Sederhana

   Graf tak sederhana mengandung sisi ganda atau gelang seperti pada gambar berikut. Graf tak sederhana dapat dibagi lagi menjadi graf ganda yang mengandung sisi ganda serta graf semu yang mengandung gelang.



Gambar 8. Graf Tak Sederhana

Graf yang tidak mempunya orientasi arah disebut graf tak berarah, sedangkan graf yang mengandung orientasi arah disebut dengan graf berarah. Pada graf berarah, terdapat sisinya yang diberikan arah dalam bentuk panah dari suatu verteks ke verteks lainnya.

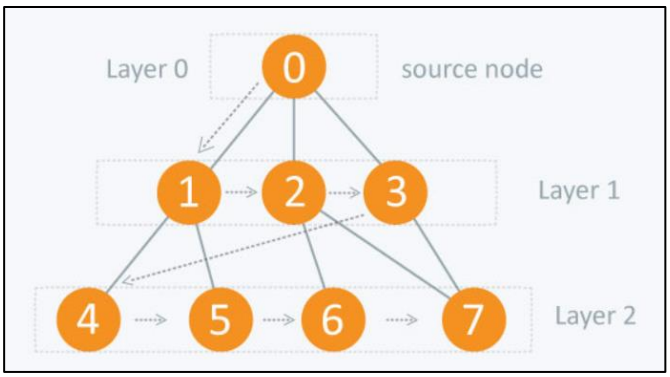


Gambar 9. Graf takberarah dan graf tak berarah

2.1.2. Breadth First Search

Algoritma Breadth First Search (BFS) termasuk ke dalam algoritma traversal graf. Pada algoritma traversal graf, setiap verteks pada graf dikunjungi secara sistematik. BFS merupakan algoritma dengan pencarian traversal melebar. Pengecekan jalur dilakukan dengan membangkitkan seluruh verteks tetangga terlebih dahulu, dilanjutkan dengan pengunjungan tetangga-tetangga tadi untuk dilakukan proses yang sama, demikian seterusnya. Adapun algoritma BFS sebagai berikut.

1. Kunjungi verteks 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.



Gambar 10. Ilustrasi Algoritma Breadth First Search

Berikut merupakan struktur data yang diperlukan dalam pemrosesan BFS.

1. Matriks ketetanggan A = [a_{ij}] yang berukuran n × n dengan a_{ij} = 1 jika simpul *i* dan simpul *j* bertetangga, sebaliknya a_{ij} = 0 jika simpul *i* dan simpul *j* tidak bertetangga.
2. Antrian q untuk menyimpan semua simpul yang telah dikunjungi
3. Tabel boolean yang menyatakan apakah elemen ke-i di tabel yang berkorelasi dengan simpul ke-i pada graf telah dikunjungi atau belum. Bernilai true apabila telah dikunjungi, sebaliknya bernilai false apabila belum dikunjungi.

2.1.3. Depth First Search

Depth First Search (DFS) juga termasuk dalam algortima pencarian traversal sebagaimana BFS. Meskipun begitu, DFS berkebalikan dengan BFS dalam hal pengecekan verteks. Pada DFS, tetangga yang dibangkitkan hanyalah satu lalu dipanggil kembali tetangga dari tetangga yang telah dipanggil sebelumnya. Berikut merupakan algoritma DFS.

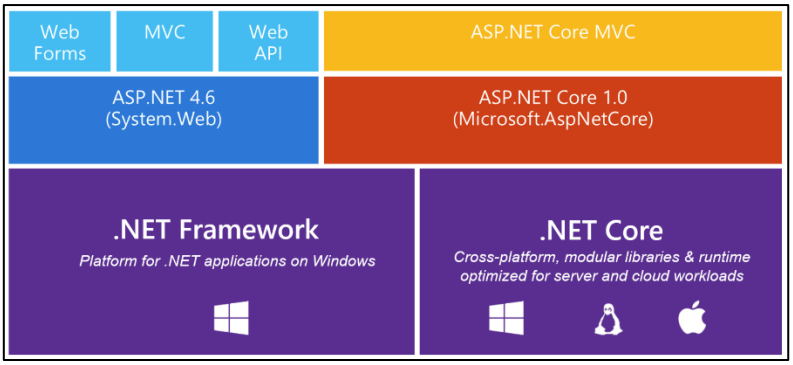
1. Kunjungi verteks v
2. Kunjungi verteks w yang bertetangga dengan verteks v
3. Ulangi DFS pada verteks w
4. Ketika mencapai verteks u sedemikan sehingga semua verteks yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunya simpul w yang belum dikunjungi
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Struktur data dari DFS mirip seperti BFS, hanya saja pada DFS tidak dibutuhkan antrian q. Jadi, pada DFS tedapat matriks ketetanggaan dan tabel boolean suatu verteks telah dikunjungi atau tidak.

2.2. C# Desktop Application Development

2.2.1. .NET Core

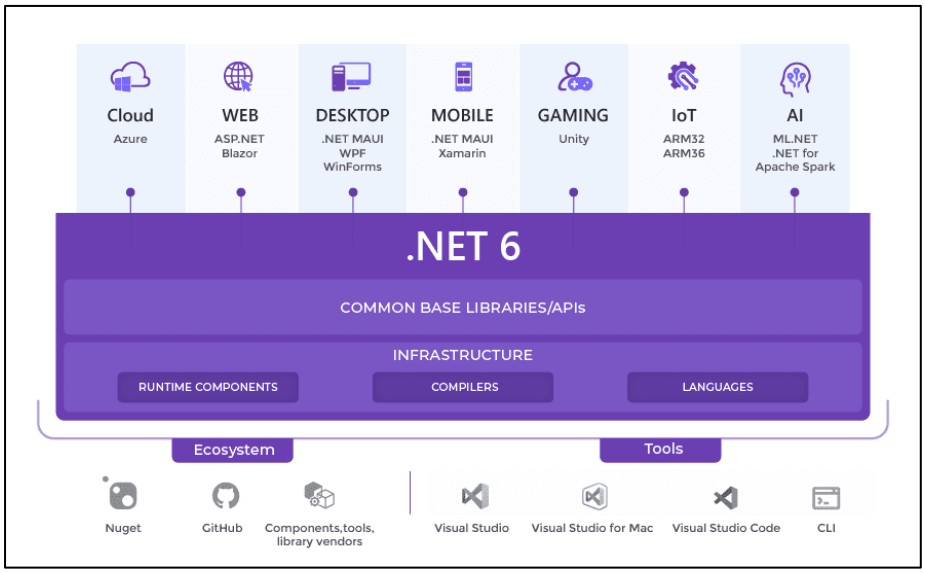
.NET Framework adalah perangkat lunak framework yang dikembangkan oleh Microsoft pada tahun 2002. Untuk membangun bermacam aplikasi, seperti aplikasi desktop, aplikasi mobile, aplikasi web dan cloud, .Net Framework dilengkapi dengan aturan kerja, aturan pemrograman, dan banyak class library. Saat ini, versi terbaru dari .NET Framework adalah versi 4.8. Aplikasi yang dikembangkan menggunakan .NET Framework hanya berjalan pada komputer bersistem operasi Microsoft Windows dan telah melakukan instalasi .NET framework. Namun, sejak tahun 2016, Microsoft mengembangkan .NET Core, yaitu .NET Framework yang bersifat open-source dan multiplatform. Ini berarti, .NET Core dapat dijalankan pada berbagai sistem operasi lain selain Windows, seperti Linux dan Mac OS. .NET Core mendukung bahasa pemrograman C# dan F#. Saat ini, versi terbaru dari .NET Core adalah versi .NET 6.0 yang merupakan versi long-term supported (LTS).



Gambar 11. Perbedaan .NET Framework dan .NET Core

2.2.2. Aplikasi Desktop

Pada Gambar 12, dapat dilihat bahwa .NET 6 dapat digunakan untuk membangun banyak tipe aplikasi, salah satunya adalah aplikasi desktop. Pada framework ini, aplikasi desktop dapat dibedakan menjadi beberapa kategori sebagai berikut.



Gambar 12. Unified Platform Support .Net 6

1. Windows Forms

   Windows Forms atau dikenal juga sebagai WinForms adalah User Interface (UI) framework untuk membangun aplikasi desktop. Secara umum, WinForms digunakan untuk membuat aplikasi desktop sebab tampila yang formal dan umum ditemui pada aplikasi desktop. Selain itu, class library WinForms telah ada sejak .NET Framework dirilis tahun 2002. Pada tahun 2018, Microsoft mengumumkan bahwa WinfForms adalah proyek open-source pada Github.

2. Windows Presentation Foundation

   Tidak seperti Windows Forms yang memberikan antarmuka yang terlihat formal dan kaku, Windows Presentation Foundation dapat memberikan antarmuka yang lebih bebas. Hal ini karena WPF menggunakan Extensible Application Markup

Language (XAML) untuk membuat antarmuka. XAML mirip seperti HTML sehingga antarmuka aplikasi WPF juga dapat didesain seperti halaman web. Kelebihan WPF yang lain adalah antarmuka yang independen dengan resolusi layar komputer dan dirender dengan format vector sehingga berapa pun besarnya resolusi yang digunakan, antarmuka tidak mengalami penurunan kualitas.

3. .NET MAUI
   .NET Multi-platform APP UI (.NET MAUI) adalah cross-platform framework untuk membuat aplikasi mobile dan desktop dengan menggunakan C# dan juga XAML. .NET MAUI dapat membuat aplikasi yang dapat berjalan di Android, iOS, macOS, dan Windows. .NET MAUI juga bersifat open-source dan merupakan evolusi dari Xamarin.Forms yang dilengkap dengan pembangunan ulang kontrol antarmuka. Meskipun .NET MAUI dapat membuat aplikasi multi-platforms dengan menggunakan satu proyek saja, .NET MAUI mampu untuk menambahkan kode tambahan untuk platform yang lebih spesifik apabila diperlukan.

2.2.3. Development Tool

Berdasarkan Gambar 12., terdapat beberapa tool yang dapat digunakan di dalam pengembangan aplikasi desktop dengan menggunakan C#, salah satunya adalah Visual Studio. Tersedia tiga pilihan lisensi pada Visual Studio, yaitu Community, Proffesional, dan Enterprise. Breikut merupakan perbandingan fitur yang dimiliki untuk masing-masing lisensi.

Supported Features	Visual Studio Community Free download	Visual Studio Professional Buy	Visual Studio Enterprise Buy
⊕ Supported Usage Scenarios	●●●○	●●●●	●●●●
Development Platform Support [2]	●●●●	●●●●	●●●●
⊕ Integrated Development Environment	●●●○	●●●○	●●●●
⊕ Advanced Debugging and Diagnostics	●●○○	●●○○	●●●●
⊕ Testing Tools	●○○○	●○○○	●●●●
⊕ Cross-platform Development	●●○○	●●○○	●●●●
⊕ Collaboration Tools and Features	●●●●	●●●●	●●●●

Gambar 13. Perbandingan Fitur Setiap Lisensi Visual Studio 2022

Kami menggunakan Visual Studio 2022 versi 17.0 dalam tugas besar ini. Visual Studio Community dipilih agar kami dapat menggunakan WPF sebagai framewok untuk pengembangan aplikasi agar tampilannya lebih modern dan fleksibel.

BAB III
ANALISA PEMECAHAN MASALAH

3.1. Langkah-langkah Pemecahan Masalah

Berdasarkan permasalahan yang terdapat pada deskripsi tugas, dijelaskan bahwa permasalahan yang perlu diselesaikan yaitu mencari file yang sesuai dengan query mulai dari starting directory hingga seluruh children dari starting directory tersebut. Struktur dari starting directory hingga seluruh children dapat direpsentasikan dengan struktur pohon.

Dalam penyelesaian permasalahan di atas, maka diperlukan pencarian node dari akar pohon hingga didapatkan node yang sesuai dengan file yang akan dicari. Pencarian node ini akan menggunakan algoritma *Breadth First Search (BFS)* atau algoritma *Depth First Search (DFS)*.

Program pada awalnya akan meminta *starting directory* dan file yang akan dicari. Setelah itu, program akan melakukan pencarian menggunakan BFS atau DFS. Untuk kasus *one occurance*, setelah suatu node sesuai dengan file yang akan dicari, maka program tidak akan melakukan pencarian kembali, sedangkan pada kasus *all occurance* pencarian akan dilanjutkan hingga seluruh *children* dari *starting directory* telah dilewati. Setelah itu, program yang telah dibuat berupa GUI akan memvisualisasikan hasil dari pencarian file dalam bentuk pohon dan list pathnya.

3.2. Proses Mapping Persoalan

3.2.1. Permasalahan One Occurence

Misalkan pencarian dilakukan dengan *starting directory* A dan file B yang akan dicari.

a) Problem state : Mencari path yang menghubungkan node A dan node B.
b) Initial state : Node A
c) Goal state : Node B
d) State space : Himpunan semua node yang ada di pohon
e) State space tree : Pohon dari state space
f) Solution space : Himpunan semua daun yang ada di pohon
g) Solution : Path yang menghubungkan node A dengan node B

3.2.2. Permasalahan All Occurence

Misalkan pencarian dilakukan dengan *starting directory* A dan file B yang akan dicari.

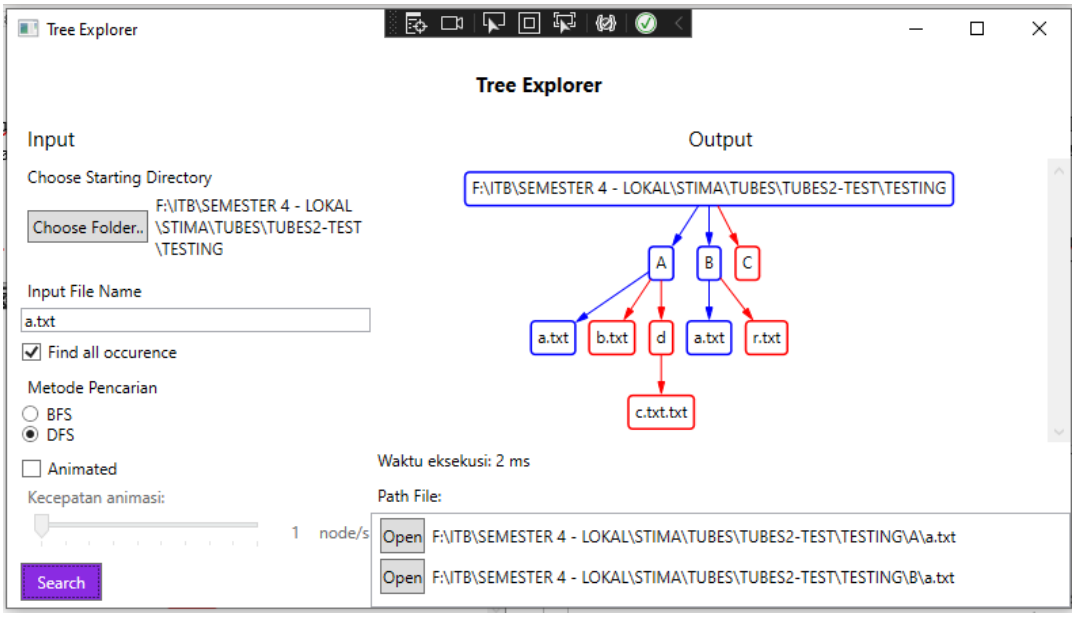
a) Problem state : Mencari satu atau lebih path yang menghubungkan node A dan node B.
b) Initial state : Node A
c) Goal state : Satu atau lebih node B
d) State space : Himpunan semua node yang ada di pohon
e) State space tree : Pohon dari state space

f) Solution space : Himpunan semua daun yang ada di pohon
g) Solution : Satu atau lebih path yang menghubungkan node A dengan node B

3.3. Ilustrasi Kasus Lain

3.3.1. Kasus DFS dengan All Occurence

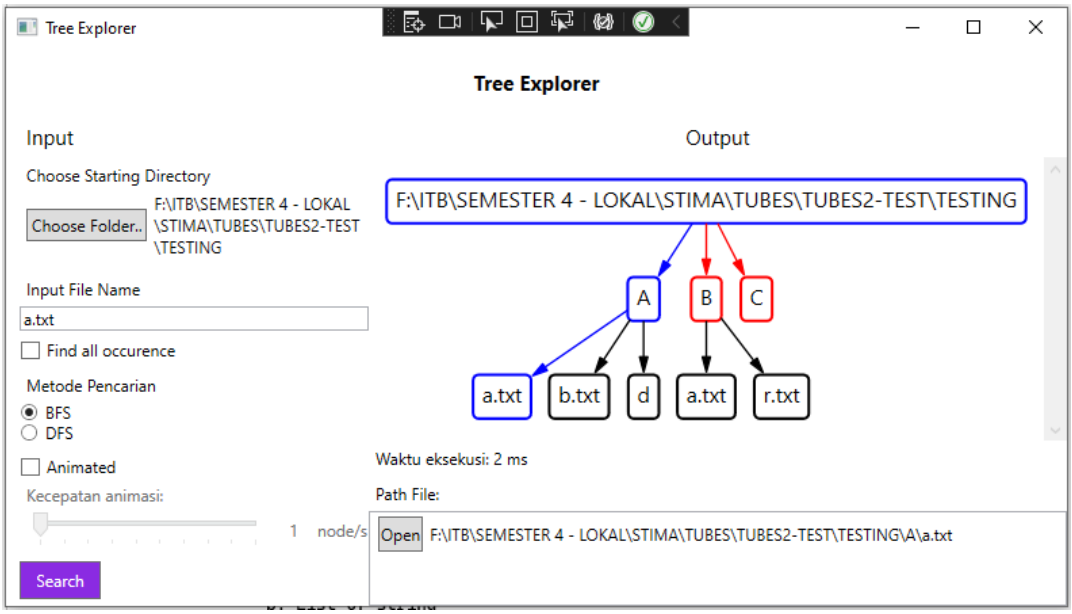
File yang akan dicari yaitu “a.txt”



Gambar 14. Output Program dengan Algoritma DFS All Occurence

3.3.2. Kasus BFS dengan One Occurence

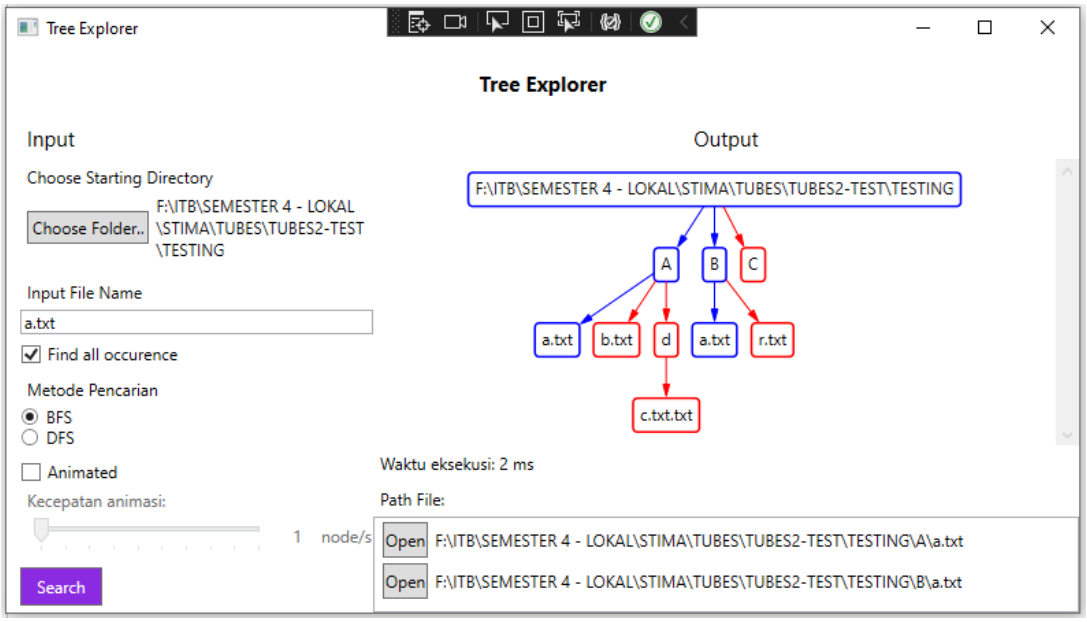
File yang akan dicari yaitu “a.txt”



Gambar 15. Output Program dengan Algoritma BFS One Occurence

3.3.3. Kasus BFS dengan All Occurrence

File yang akan dicari yaitu “a.txt”



Gambar 16. Output Program dengan Algoritma BFS All Occurence

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Program

Algoritma utama yang digunakan dalam program ini adalah BFS dan DFS. Berikut merupakan implementasi dari class BFS dan DFS dengan menggunakan pseudocode.

4.1.1. Implementasi BFS

Atribut:

```
searchedFile: string
startingTree: Tree
isOneOccurance: boolean
resultPaths: List of string
path: List of string
queue: Queue of Tree
childPath: Queue of List of string
```

Constructor:

```
BFS(input searchedFile: string, input startingTree: Tree, input:
isOneOccurance: boolean)
```

Kamus:

```
p: List of string
```

Algoritma

```
this.startingTree = startingTree
this.searchedFile = searchedFile
this.isOneOccurance = isOneOccurance
initial(startingTree)
searchFile(this.startingTree)
```

Method:

```
1.  procedure searchFile(input tree: Tree, input/ouput path: List of
    string)
```

Kamus:

```
temp: Tree
```

Algoritma:

```
if (isOneOccurance)
      if (not tree.info = searchedFile or not tree.isLeaf()) then
            BFSRecursive(tree)
            if (queue.Count > 0) then
                  searchFile(queue.Peek())
      tree.changeToRED();
      if (tree.info = searchedFile and tree.isLeaf()) then
            treeColoring(childPath.Peek())
            while (queue.Count > 0) do
                  temp = queue.Dequeue()
                  temp.children.Clear()
else
      if (not tree.info = searchedFile or not tree.isLeaf()) then
            BFSRecursive(tree)
            if (queue.Count > 0) then
```

```
                searchFile(queue.Peek())
        tree.changeToRED()
        if (tree.info = searchedFile and tree.isLeaf()) then
              treeColoring(childPath.Peek())
              queue.Dequeue()
              path = childPath.Dequeue()
              if (queue.Count > 0) then
                    searchFile(queue.Peek())
```

2. `procedure BFSRecursive(input tree: Tree)`

   Kamus:

   ```
   newP: List of string
   ```

   Algoritma:

   ```
   queue.Dequeue()
   path = childPath.Dequeue()
   foreach (Tree childRecursive in tree.children)
         queue.Enqueue(childRecursive)
         path.Add(childRecursive.info)
         childPath.Enqueue(newP)
         path.Remove(childRecursive.info)
   ```

3. `procedure Initial(input tree: Tree)`

   Kamus:

   ```
   path: List of string
   queue: Queue of Tree
   childPath: Queue of List of string
   currentPath: List of string
   ```

   Algoritma:

   ```
   currentPath.Add(tree.info)
   queue.Enqueue(tree)
   childPath.Enqueue(currentPath)
   ```

4. `procedure treeColoring(input path: List of string)`

   Kamus:

   ```
   nodeTree: Tree
   cumulPath: string
   i: integer
   nextNode: Tree
   ```

   Algoritma:

   ```
   nodeTree ← startingTree
   cumulPath ← “”
   i ← 1
   while (not nodeTree.isLeaf()) do
         nodeTree.changeToBLUE()
         cumulPath ← cumulPath + nodeTree.info + "\\"
               nextNode ← nodeTree.getChild(path[i]);
               i ← i + 1
                if (nextNode ≠ null) then
                         nodeTree ← nextNode
   nodeTree.changeToBLUE()
   cumulPath ← cumulPath + nodeTree.info
   resultPaths.Add(cumulPath)
   ```

4.1.2. Implementasi DFS

Atribut:

```
searchedFile: string
startingTree: Tree
isOneOccurance: boolean
totalOccurance: integer
resultPaths: List of string
```

Constructor:

```
DFS(input searchedFile: string, input startingTree: Tree, input:
isOneOccurance: boolean)
```

Kamus:

```
p: List of string
```

Algoritma

```
this.startingTree = startingTree
this.searchedFile = searchedFile
this.isOneOccurance = isOneOccurance
this.totalOccurance = 0
p = [this.startingTree.info]
searchFile(this.startingTree, p)
```

Method:

1. `procedure searchFile(input tree: Tree, input/ouput path: List of string)`

Kamus:

```
newP: List of string
```

Algoritma:

```
if (not (isOneOccurance and totalOccurance = 1) then
      if (tree.info ≠ startingTree.info) then
            path.Add(tree.info)
      tree.changeToRED()
      if (tree.info = searchedFIle and tree.isLeaf()) then
            treeColoring(path)
            totalOccurance ← totalOccurance + 1
if (not tree.isLeaf) then
            foreach (child in tree.children)
                  if (isOneOccurance and totalOccurance = 1) then
                        child.children.Clear()
                  else
                        newP ← copy isi path
                        searchFile(child, newP)
```

2. `procedure treeColoring(input path: List of string)`

Kamus:

```
nodeTree: Tree
cumulPath: string
i: integer
nextNode: Tree
```

Algoritma:

```
nodeTree ← startingTree
cumulPath ← “”
```

```
i ← 1
while (not nodeTree.isLeaf()) do
      nodeTree.changeToBLUE()
      cumulPath ← cumulPath + nodeTree.info + "\\"
           nextNode ← nodeTree.getChild(path[i]);
           i ← i + 1
            if (nextNode ≠ null) then
                    nodeTree ← nextNode
nodeTree.changeToBLUE()
cumulPath ← cumulPath + nodeTree.info
resultPaths.Add(cumulPath)
```

4.2. Struktur Data dan Spesifikasi Program

4.2.1. Struktur Data

Struktur data yang digunakan untuk menyimpan informasi mengenai folder atau file adalah Tree, sedangkan struktur data yang digunakan untuk menampilkan hasil pencarian adalah Graph. Struktur data Graph yang digunakan berasal dari built-in Microsoft.Msagl.Drawing. Adapun struktur data Tree yang digunakan terdapat dalam file Tree.cs yang kami buat sendiri. Berikut merupakan implementasi dari strukur data Tree.

Atribut:

```
info: string
{ berisi nama file atau folder }
nodeColor: TreeColor
{ TreeColor dapat berupa BLACK, RED, BLUE }
children: Lists of Tree
{ berisi Tree lainnya apabila Tree ini merupakan folder }
```

Metode:

```
Tree()
{ constructor }
Tree(info: string)
{ user-defined constructor, default color = BLACK}
Tree(info: string, nodeColor: TreeColor)
{ user-defined constructor }
function isLeaf() → boolean
{ true apabila tidak punya anak alias Tree ini adalah file }
procedure changeToRED()
{ ubah nodeColor menjadi RED apabila sebelumnya warna adalah BLACK}
procedure changeToBLUE()
{ ubah nodeColor menjadi BLUE }
function getChild(nodeInfo: string) → Tree
{ mengembalikan sebuah tree dari list children apabila info dari
children sama dengan nodeInfo }
```

Selain struktur data Tree, di dalam file Tree.cs terdapat pula beberapa struktur data lain yang dibuat, yaitu sebagai berikut.

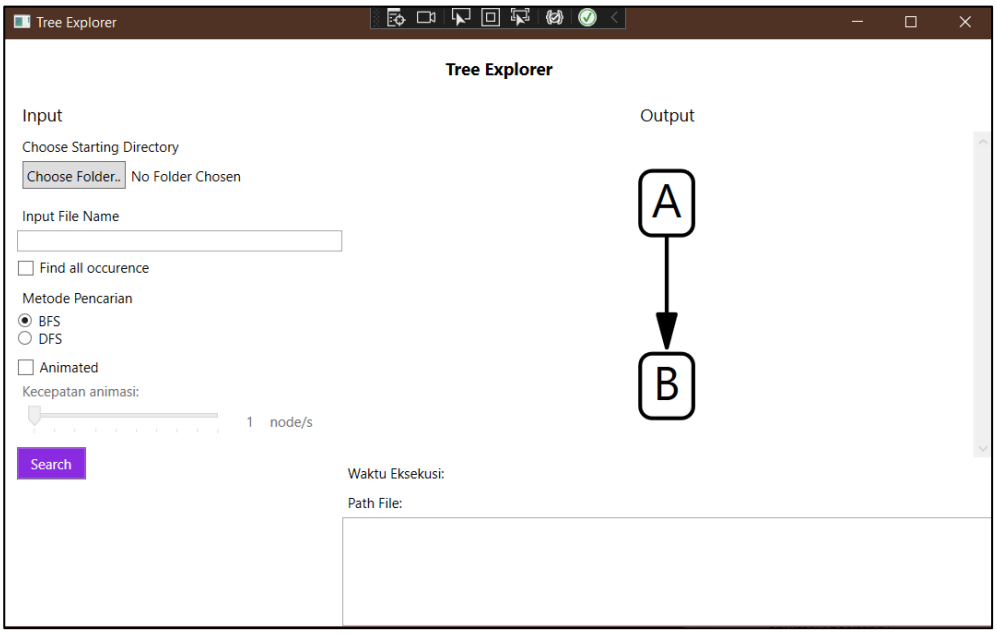
1. `enum TreeColor { BLACK, RED, BLUE }`
2. `enum MetodeSearch { BFS, DFS }`

3. `class TreeColorToRGB`

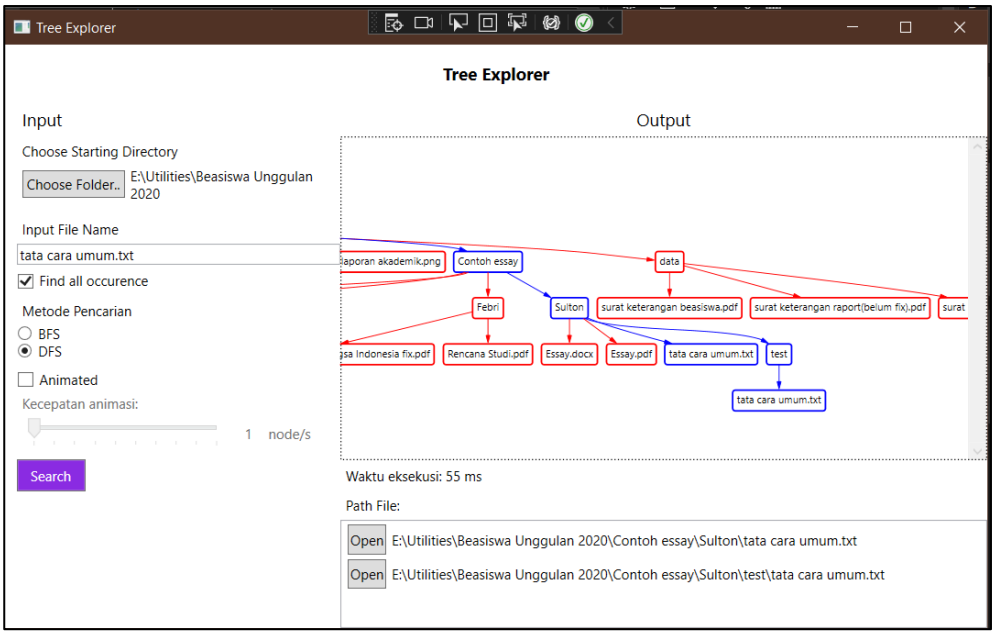
4.2.2. Spesifikasi Program

File MainWindow.xaml bertanggung jawab untuk bagian antarmuka program (frontend), sedangkan file MainWindow.xaml.cs bertanggung jawab untuk bagian logika aplikasi (backend).

Berikut merupakan tampilan dari aplikasi kami ketika dijalankan untuk pertama kali dan ketika program telah berhasil mencari file.



Gambar 17. Aplikasi Dijalankan untuk Pertama kali



Gambar 18. Aplikasi Berhasil Menemukan File yang Dicari

Berikut merupakan spesifikasi dari GUI:

1. Program dapat menerima input folder dan 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. 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.

Adapun spesifikasi dari mainWindow yang terdapat di dalam file mainWindow.xaml.cs adalah sebagai berikut.

Atribut:

```
dirTree: Tree
{ berisi nama direktori awal dilakukannya pencarian }
graph: Graph
dfsanim: DFSAnimated
bfsanim: BFSAnimated
stopwatch: Stopwatch
```

Metode:

```
MainWindow()
{ constructor }
function cloneGraph(graph: Graph) → Graph
{ Membuat salinan graph }
procedure updateGraphControl(input graph: Graph)
{ Mengupdate visualisasi graph yang ada di Thread UI/utama }
procedure notifyAnimatedFinished()
{ Menginformasikan dari thread anak/animasi ke Thread UI/utama bahwa
    animasi telah selesai }
procedure chooseFolderButton_Click()
{ Membuka menu pemilihan folder }
procedure updateGraph(input path: string, input tree: Tree)
{ Mengupdate visualisasi graph berdasarkan tree masukan }
procedure constructDirectoryTree(input path: string)
{ Membuat pohon direktori dari path folder masukan }
procedure searchButton_Click(input sender: object, input e:
RoutedEventArgs)
{ Memulai pencarian }
procedure skipButton_Click(input sender: object, input e:
RoutedEventArgs)
{ Pada penelusuran dengan animasi, animasi dapat dilewati langsung pada
    hasil akhir }
```

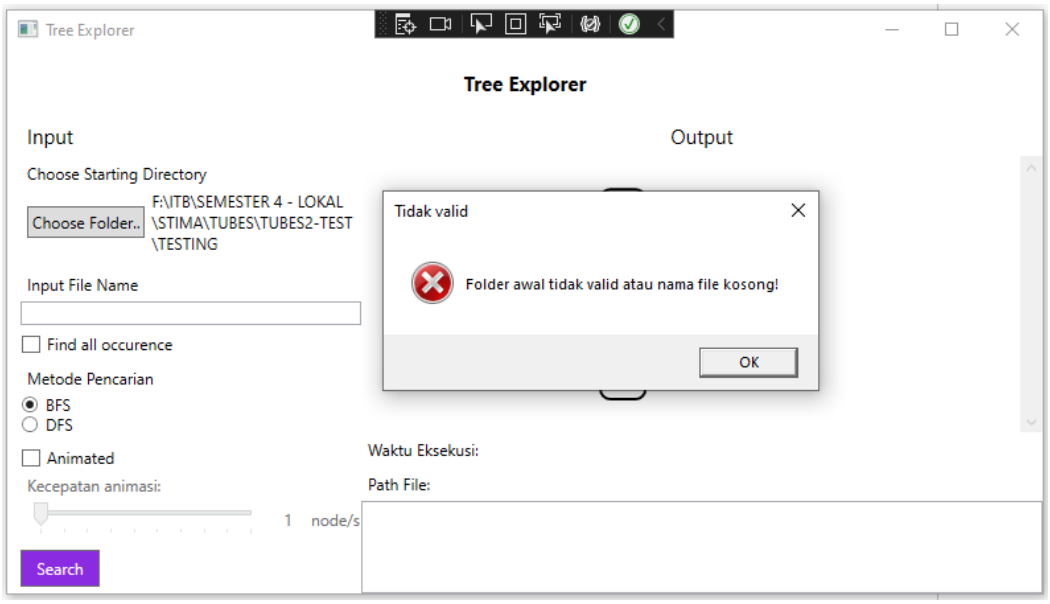
4.3. Tata Cara Penggunaan Program

- Buka file Tree Explorer.exe yang berada di dalam folder bin
- Saat aplikasi telah terbuka, akan ada dua bagian, yaitu input dan output.
- Pada bagian input, tekan tombol `Choose Folder` untuk memilih direktori awal dilakukannya pencarian suatu file
- Pada bagian kolom `Input File Name` isi dengan nama file yang ingin dicari disertai dengan ekstensinya
- Checkboxes `Find all occurence` dapat dicentang apabila pengguna menginginkan pencarian untuk semua file yang memiliki nama sama dengan `input File Name`. Apabila dibiarkan tidak tercentang, pencarian akan berhenti ketika menemukan satu saja file yang bernama sama dengan `input File Name`
- Pada bagian Metode Pencarian, pengguna dapat memilih metode pencarian yang akan digunakan untuk mencari file di dalam direktori. Tersedia metode BFS dan DFS yang dapat dipilih oleh pengguna.
- Checkboxes `Animated` dapat dicentang apabila pengguna menginginkan output pencarian dilakukan secara bertahap (dengan animasi) sesuai dengan metode pencarian.
- Apabila `Animate` dicentang, pengguna dapat mengatur kecepatan munculnya setiap file atau folder pada bagian output dengan menggeser slider.
- Setelah semua input telah terpenuhi, tombol `Search` dapat diklik untuk memunculkan output pencarian.
- Pada Gambar 18. dapat terlihat hasil output pencarian. Warna hitam pada node berarti node tersebut telah dibangkitkan, tetapi tidak dilakukan pemeriksaan. Warna merah menunjukkan node yang telah diperiksa, sedangkan warna biru menunjukkan jalur mulai dari direktori awal sampai file yang dicari ditemukan.
- Selain itu, terdapat pula waktu eksekusi dan path file yang dicari yang dapat dibuka dengan mengeklik tombol `open`.

4.4. Hasil Pengujian

4.4.1. Pengujian Saat Input File atau Starting Directory Kosong

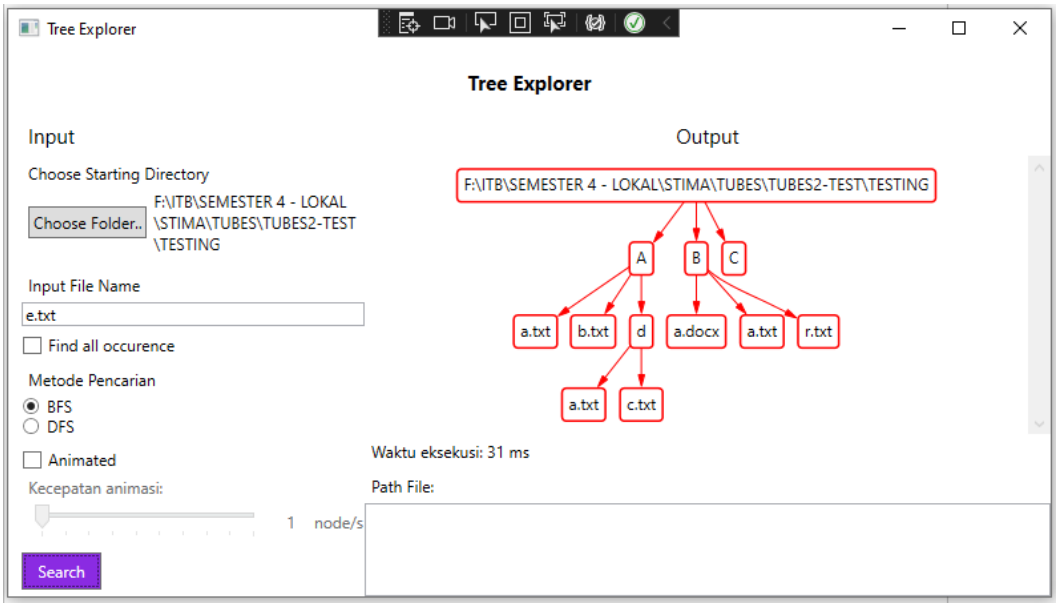
Ketika input file atau starting directory yang diperlukan oleh program tidak diinputkan oleh pengguna, maka program akan mengeluarkan pesan kesalahan.



Gambar 19. Output Program ketika Input File atau Starting Directory Kosong

4.4.2. Pengujian Saat File Tidak Ditemukan

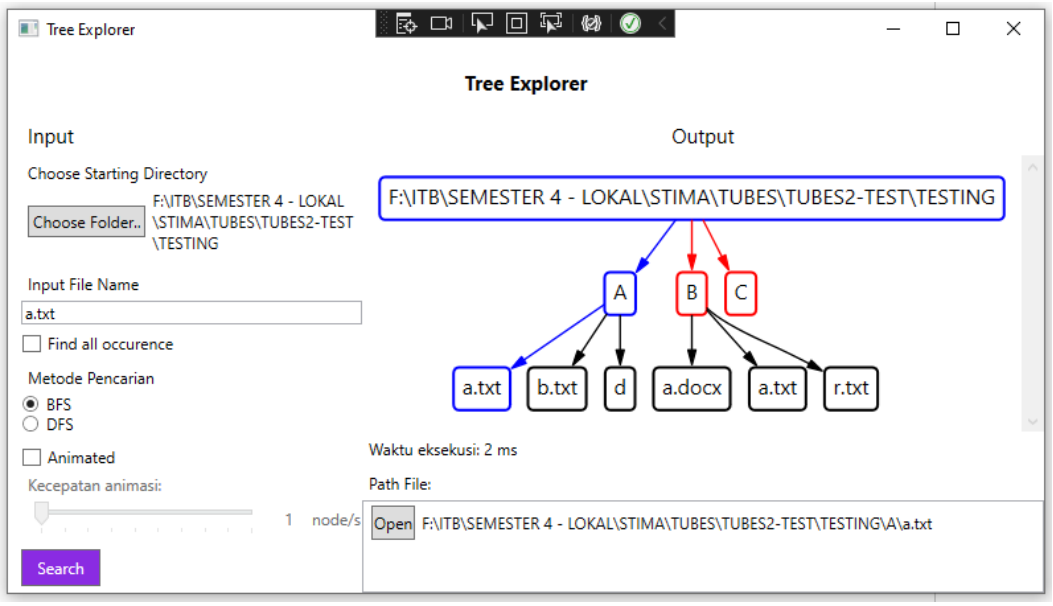
Ketika input file yang ingin dicari tidak terdapat pada direktori, maka node yang sudah dilewati dan bukan merupakan file yang ingin dicari akan berwarna merah.



Gambar 20. Output Program ketika file tidak ditemukan

4.4.3. Pengujian dengan One Occurence

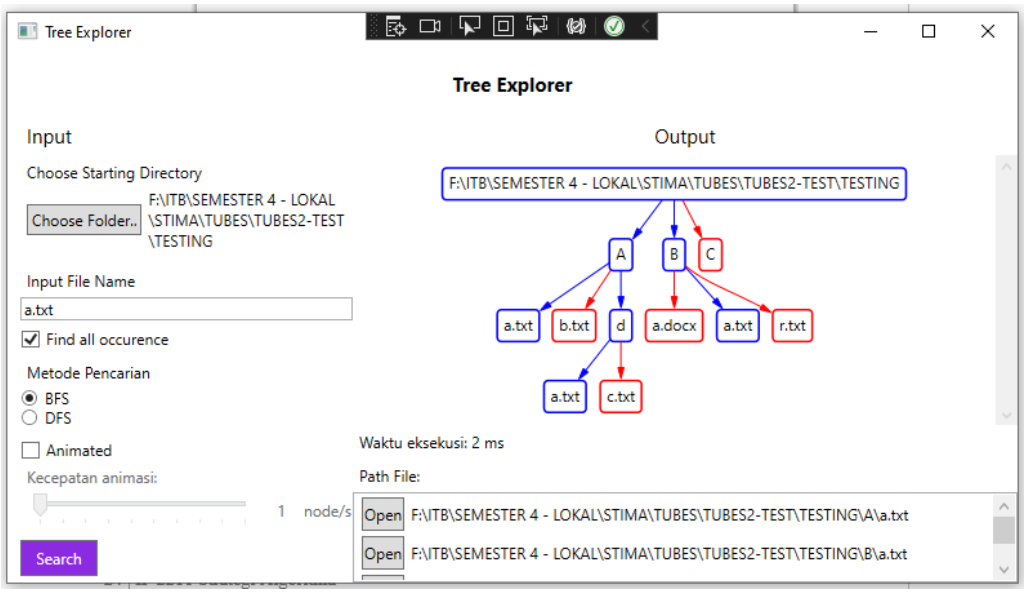
Pengujian dilakukan dengan algoritma BFS. Ketika proses pencarian dilakukan dan ditemukan node yang sesuai dengan Input File maka proses pencarian akan berhenti dan memvisualisasikan hasil pencarian.



Gambar 21. Output Program ketika file ditemukan dengan one occurence

4.4.4. Pengujian dengan All Occurence

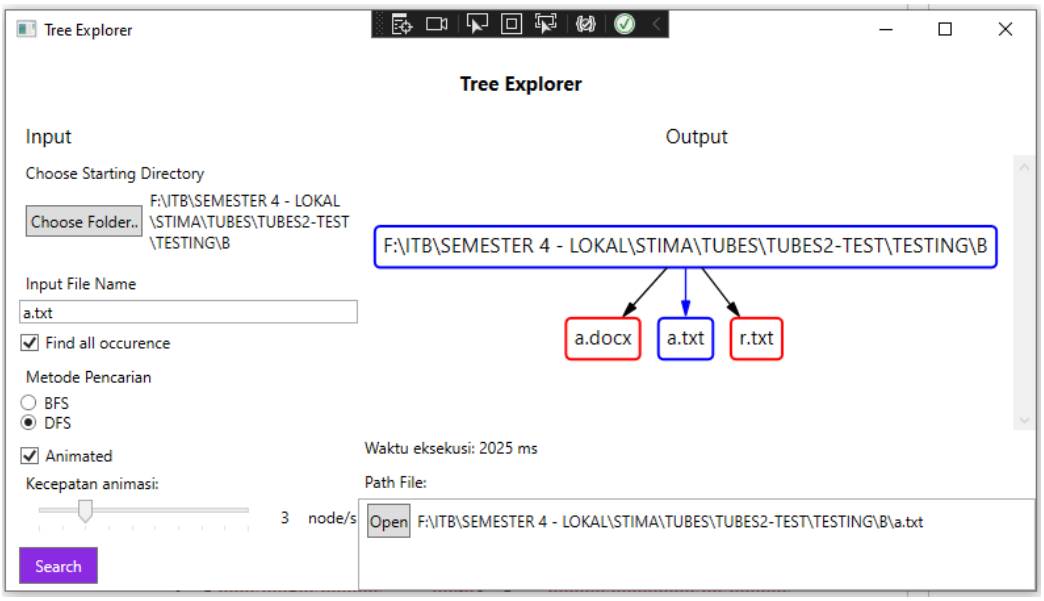
Pengujian dilakukan dengan algoritma BFS. Berbeda halnya dengan one occurence, pencarian all occurence akan tetap dilanjutkan ketika node yang sedang dicek sesuai dengan Input File hingga seluruh anak dari akarnya telah dicek.



Gambar 22. Output Program ketika file ditemukan dengan all occurence

4.4.5. Pengujian Saat File yang Sama Berbeda Ekstensi

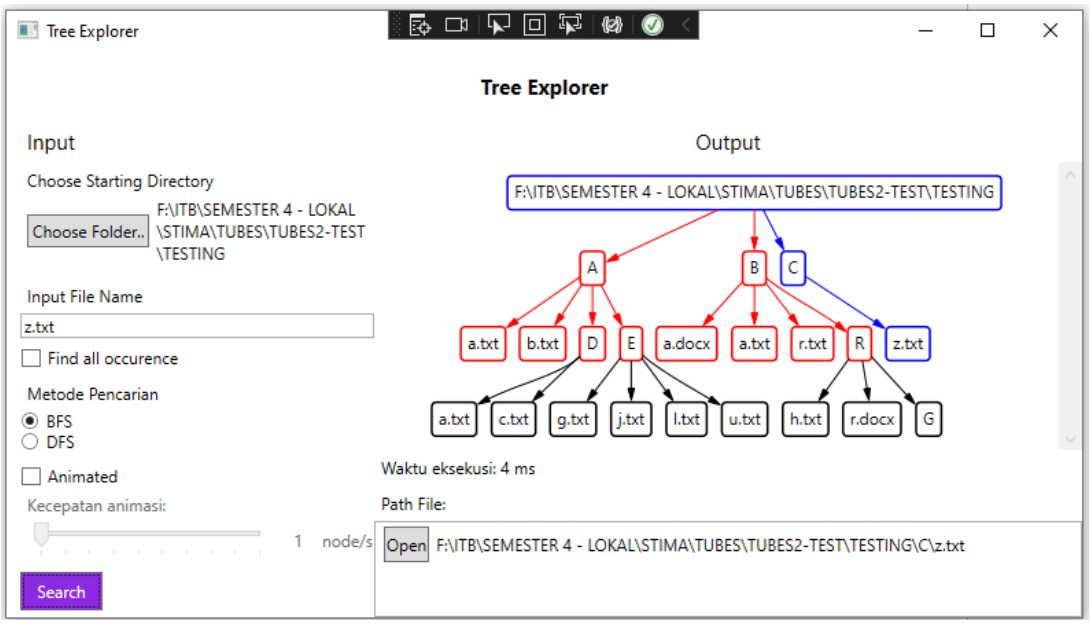
Pencarian dilakukan dengan algoritma DFS. Ketika di dalam folder yang sama terdapat satu file yang sama tetapi berbeda ekstensinya, maka program akan menentukan apakah node sesuasi dengan Input File atau tidak. Hal ini dapat diliat ketika file yang ingin dicari yaitu “a.txt” tetapi dalam folder B terdapat nama file yang sama dengan ekstensi docx, tetapi program berhasil mendeteksi hal tersebut.



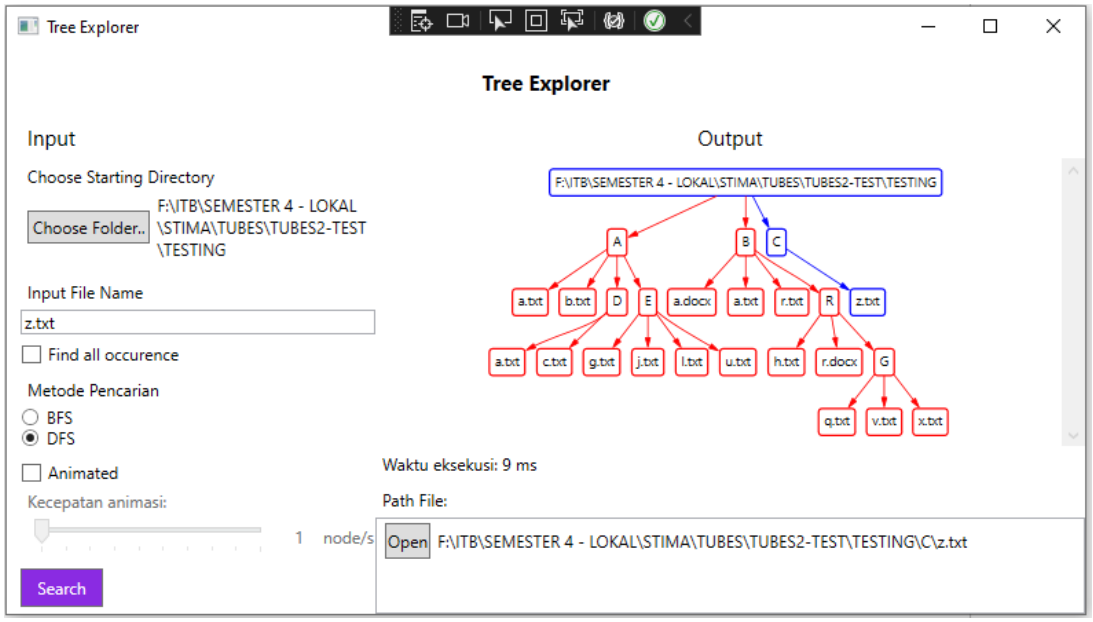
Gambar 23. Output Program saat file yang sama tetapi berbeda ekstensi

4.5. Analisis Desain Solusi

Proses pencarian dengan algoritma BFS dan DFS memiliki keuntungannya masing-masing yang berbeda. Pada algoritma pencarian BFS, pencarian akan dilakukan pada semua node pada tiap levelnya secara berurututan dari kiri ke kanan. Jika pada suatu level belum ditemukan node yang akan dicari, maka akan pencarian dilakukan pada level berikutnya hingga ditemukan solusi. Hal ini akan menghasilkan waktu pencarian lebih cepat dibandingkan dengan algoritma DFS jika node yang akan dicari terdapat pada level yang dekat dengan akar dan paling kanan.

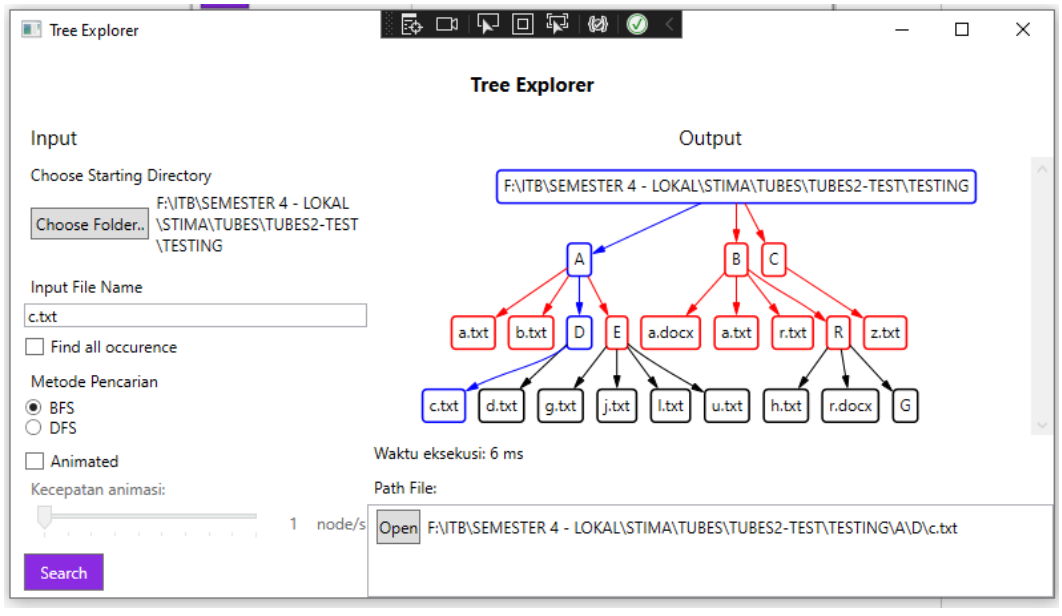


Gambar 24. Waktu eksekusi dengan algoritma BFS (4 ms)

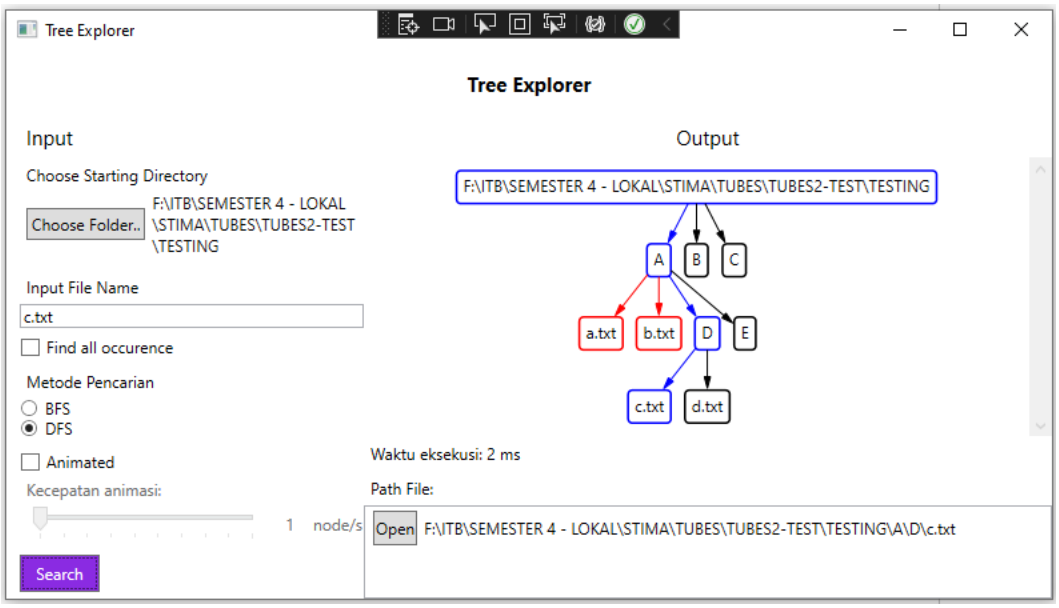


Gambar 25. Waktu eksekusi dengan algoritma DFS (9 ms)

Sedangkan pada algoritma DFS, pencarian dilakukan pada suatu kedalaman node yang paling kiri dan masih bisa ditelusuri. Jika pada level terdalamnya masih tidak ditemukan solusi, maka pencarian akan dilakukan pada level sebelumnya hingga ditemukan solusi. Hal ini akan menghasilkan waktu yang lebih cepat dibandingkan dengan algoritma BFS jika node yang akan dicari terdapat pada level yang terdalam dan paling kiri.



Gambar 26. Waktu eksekusi dengan algoritma BFS (6 ms)



Gambar 27. Waktu eksekusi dengan algoritma DFS (2 ms)

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Algoritma pencarian graf dapat dilakukan dengan beberapa metode, di antaranya adalah metode Breadth First Search dan metode Depth First Search. Metode pencarian ini dapat diterapkan untuk melakukan folder crawling. Folder crawling melakukan pencarian file yang sesuai dengan query dimulai dari starting directory hingga seluruh children dari starting directory tersebut sampai satu file pertama atau seluruh file ditemukan atau tidak ada file yang ditemukan. Penerapaan folder crawling dapat dibuat menjadi aplikasi desktop menggunakan WPF yang dikembangkan menggunakan Visual Studio. Dalam aplikasi ini digunakan struktur data Tree dan Graph. Metode BFS terkadang lebih optimal daripada DFS untuk kasus tertentu, begitu pula sebaliknya di kasus lain, metode DFS terkadang lebih baik daripada BFS.

5.2. Saran

1. Program dapat ditambah dengan metode pencarian lain yang lebih variatif, seperti Depth-Limited Search (DLS) dan Iterative Deepening Search (IDS).
2. Program dapat ditambah dengan opsi untuk mematikan visualisasi graf dan hanya menampilkan folder hasil karena visualisasi graf cukup memakan waktu dan sumber daya.

DAFTAR LINK

Link Github :
https://github.com/mhilmirinaldi/Tubes2-Stima

Link Video :
https://youtu.be/7sBR3UEBN-4

DAFTAR PUSTAKA

[1] https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui diakses tanggal 23 Maret 2022
[2] https://www.rishabhsoft.com/blog/dotnet-6-features diakses tanggal 23 Maret 2022
[3] https://www.researchgate.net/publication/340133969_Seri_Belajar_Windows_Forms_Membangun_Aplikasi_Desktop_berbasis_NET_Core_31_dengan_Visual_Studio_2019 diakses tanggal 23 Maret 2022
[4] https://thienn.com/aspnet-vs-aspnetcore/ diakses tanggal 23 Maret 2022
[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Makalah2021/Makalah-Stima-2021-K2%20(14).pdf diakses tanggal 23 Maret 2022
[6] https://www.c-sharpcorner.com/article/getting-started-with-net-6-0/ diakses tanggal 23 Maret 2022