Laporan Tugas Besar 2 IF2211 Strategi Algoritma
Semester II tahun 2021/2022

Pemanfaatan Algoritma BFS dan DFS dalam Implementasi Folder Crawling

Dipersiapkan oleh:
Kelompok 45: My Tubes My Adventure
13520047 Hana Fathiyah
13520053 Yohana Golkaria Nainggolan
13520110 Farrel Ahmad

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

DAFTAR ISI

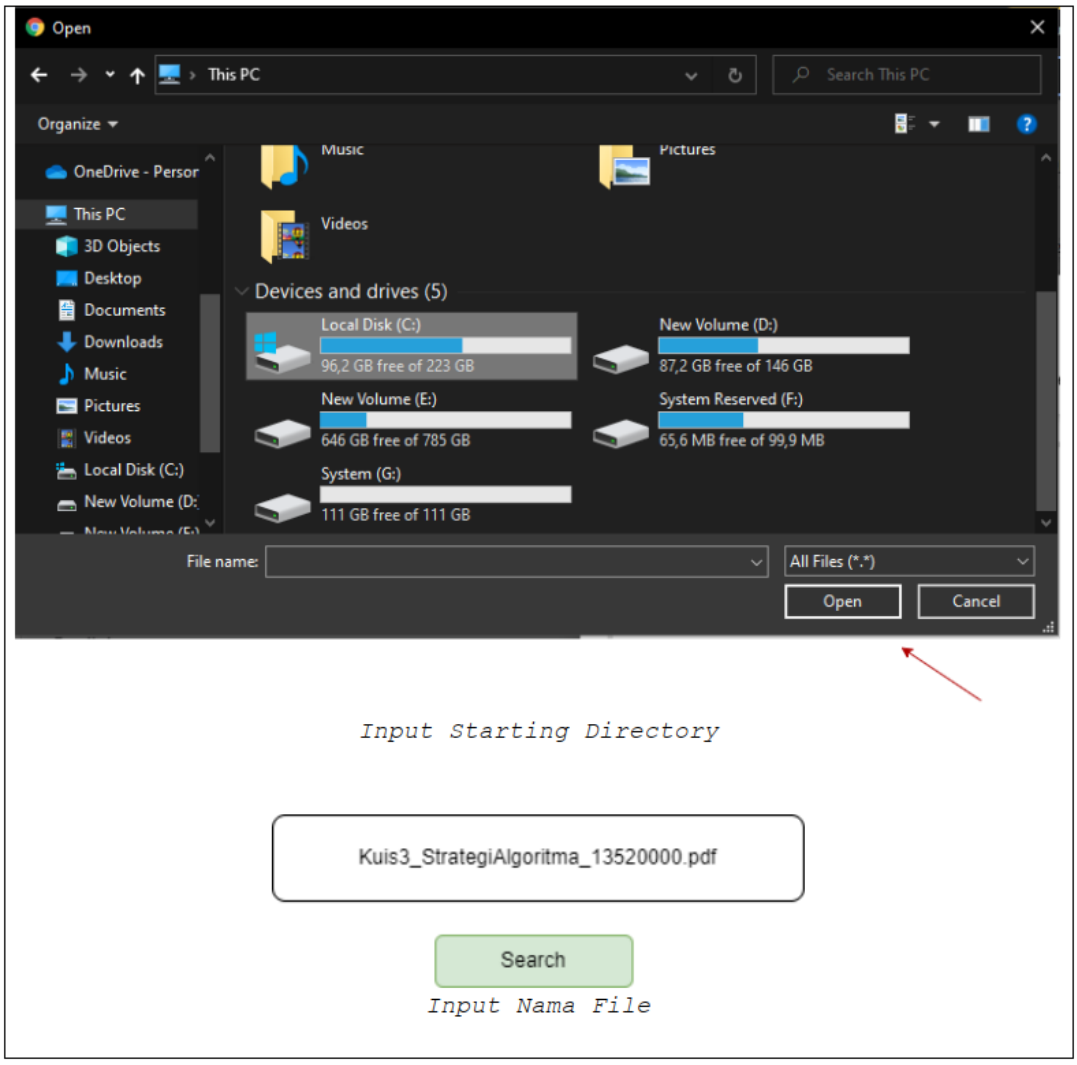
BAB I DESKRIPSI TUGAS

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.

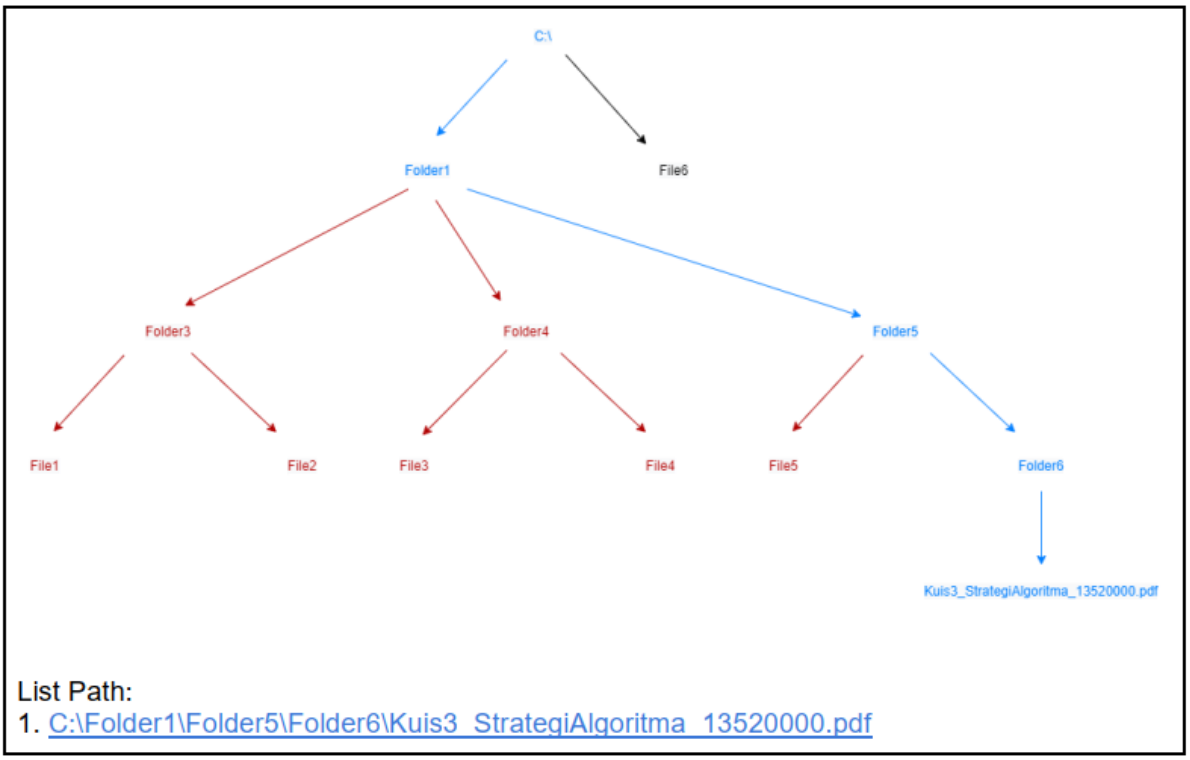
Contoh Input dan Output Program

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.

BAB II LANDASAN TEORI

2.1 Dasar Teori (Graf traversal, DFS, BFS)

2.1.1 Graf Traversal

Graf adalah representasi persoalan. Graf traversal merupakan graf pemcarian solusi. Traversal dalam graf berarti mengunjungi simpul-simpul secara sistematik, baik dengan melakukan pencarian melebar (Breadth First Search/BFS) maupun dengan pencarian mendalam (Depth First Search/DFS) dengan asumsi graf terhubung.

Algoritma graf traversal dapat dibedakan menjadi dua berdasarkan informasi yang diberikan, yaitu tanpa informasi (uninformed/blind search) dimana tidak terdapat informasi tambahan terkait pencarian solusi. Contoh : BFS, DFS, Depth Limited Search, Iterative Deepening Search, Uniform Cost Search serta dengan informasi (informed search) dimana pencarian dilakukan berbasis heuristic dan mengetahui non-goal state. Contoh: Best First Search, A*.

Dalam proses pencarian solusi, terdapat dua pendekatan:

1. Graf statis: graf yang sudah terbentuk sebelum proses pencarian dilakukan. Di sini, graf direpresentasikan sebagai struktur data
2. Graf dinamis: graf yang terbentuk saat proses pencarian dilakukan. Di sini, graf tidak tersedia sebelum pencarian, graf dibangun selama pencarian solusi.

2.1.2 DFS (Depth First Search)

DFS (Depth First Search) merupakan proses traversal dengan metode pencarian mendalam. Traversal dimulai dari simpul v yang merupakan root dari graf.

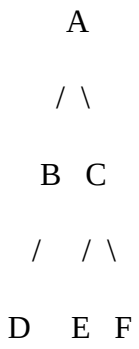
Algoritma :

1. Kunjungi simpul v,
2. Kunjungi simpul w yang bertetangga dengan simpul v.
3. Ulangi DFS mulai dari simpul w.
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik

(backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.

5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Ilustrasi dari DFS:



Urutan simpul-simpul yang dikunjungi : A – B – D – C – E – F

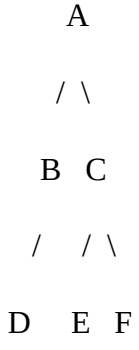
2.1.3 BFS (Breadth First Search)

Breadth First Search merupakan proses traversal dengan metode pencarian melebar. Traversal dimulai dari simpul v yang merupakan root dari graf.

Algoritma:

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

Ilustrasi BFS:

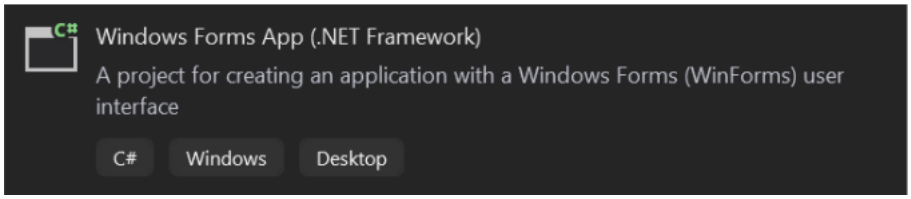


Urutan simpul-simpul yang dikunjungi : A – B – C – D – E – F

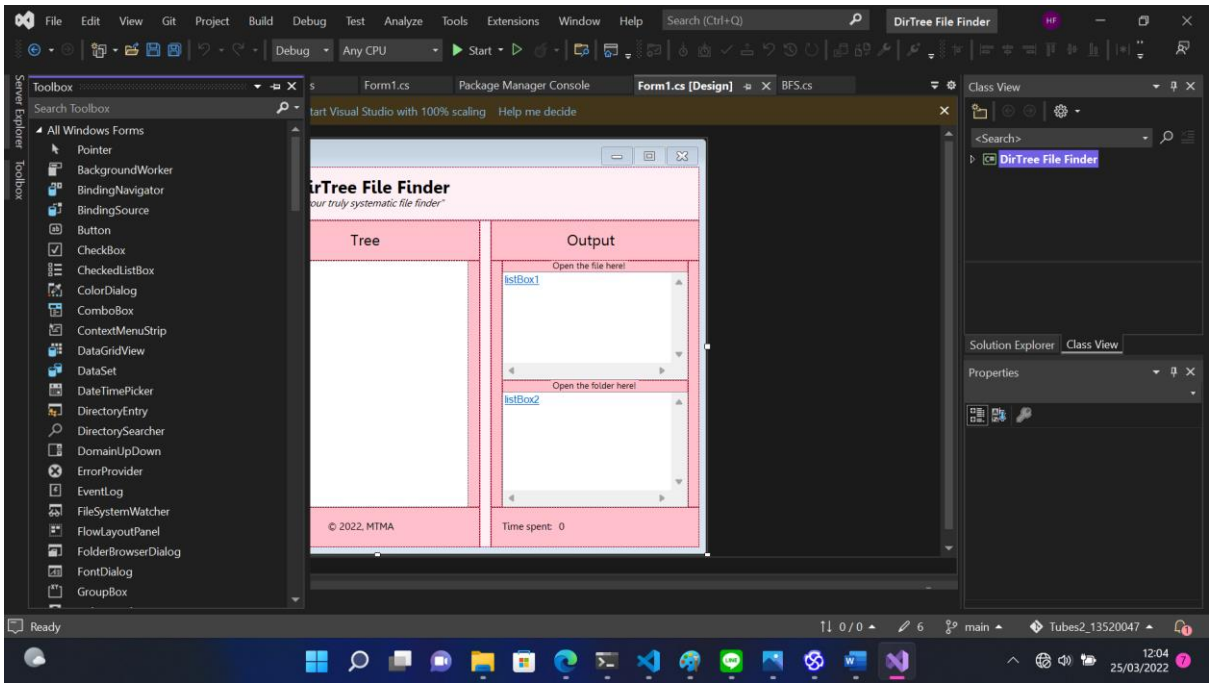
2.2 C# Desktop Application Development

Pada tugas besar ini digunakan framework Windows Form App Project .Net. Windows Form App dibuat pada Windows Graphics Device Interface (GDI) Engine. Pada pembuatan tugas ini dibuat GUI dan Algoritma secara paralel lalu kedua hal tersebut diintegrasikan pada akhir pembuatan.

Program ini memanfaatkan kakas .NET untuk C#. Direkomendasikan template Windows Form App untuk aplikasi yang memanfaatkan GUI.



Selanjutnya ada beberapa fitur di dalam Visual Studio. Di sebelah kiri terdapat *toolbox* untuk menambahkan komponen pada GUI. Di sebelah kanan terdapat *class view*. Untuk menambahkan kelas baru, dapat melakukan klik kanan dan *add new class*.



Untuk operasi interaksi antarkomponen pada GUI, digunakan Bahasa C#. Penggunaan tersebut melibatkan konsep *Object-Oriented Programming*.

Apabila program hendak dijalankan terdapat tombol *start* di bagian atas dan program akan langsung di-*debugging* oleh sistem. Penjelasan lebih lengkap terdapat pada tautan berikut ini. C# docs - get started, tutorials, reference. | Microsoft Docs

BAB III ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Pada mulanya, aplikasi dibuat dengan menggunakan Windows Forms App .NET yang dapat dengan mudah digunakan di dalam Visual Studio. Program dibuat dengan memanfaatkan GUI yang tersedia di dalam *toolbox* pada aplikasi Visual Studio. Tahap pertama yang dilakukan adalah membuat desain pengguna dengan beberapa komponen di-*rename* agar mempermudah pada saat penggabungan proses dengan tampilan antarmuka.

Dibuat suatu `Base_Class_Searcher.cs` guna menyamakan format dari proses pencarian. Setelah itu, dengan menggunakan konsep *inheritance*, dibuat kelas BFS dan DFS yang merupakan *child class* dari *parent class* `Base_Class_Searcher.cs`. Semua kelas tersebut dioperasikan di dalam file `Form.cs` guna mengintegrasikan antara proses di belakang layer dengan tampilan antarmuka.

Setelah itu, dibuat kelas `Tree.cs` untuk menampilkan tree dengan memanfaatkan MSAGL yang dapat diinstall melalui *NuGet Pacakge Manager*. Pengimplementasian *tree* ini dilakukan dengan menggunakan parameter berupa `search_log`, `foundFilePath`, dan `uncheckedPath` yang masing-masing merupakan atribut pada setiap kelas BFS dan DFS. Setelah *tree* berhasil dibuat, dilakukan integrasi dengan *tools* GViewer yang terdapat di dalam *toolbox* pada aplikasi Visual Studio dengan menggunakan *delegate* untuk mencegah terjadinya *deadlock* pada saat aplikasi tersebut dijalankan.

3.2 Proses Mapping Persoalan Menjadi Persoalan BFS DFS

Persoalan pencarian file ini dapat dimodelkan menjadi persoalan BFS dan DFS. Oleh karena proses ini merupakan proses pencarian, sebuah file atau folder dapat dimodelkan sebagai simpul dalam BFS dan DFS, sedangkan sisi pada tree (pohon) menandakan hubungan directory dengan file atau directory dengan directory lainnya. Misalkan, apabila ada simpul yang bersisian dengan simpul lain, simpul parent merupakan directory dan simpul child adalah isi dari simpul parent tersebut. Proses pencarian file ini dapat traversal BFS dan DFS. Keduanya memiliki cara masing-masing dalam upaya menemukan file yang dicari. Pada implementasi program ini *tree* yang digunakan berupa graf dinamis karena keseluruhan graf dari isi folder belum terbentuk dan akan terbentuk seiring berjalannya traversal.

3.3 Contoh Ilustrasi Kasus Lain

Algoritma BFS dan DFS juga sering digunakan untuk proses pencarian jalan (*path planning*). Baik menggunakan graf statis ataupun dinamis, BFS dan DFS dapat mencari jalan dengan traversal dari simpul awal (*start node*) menuju simpul tujuan (*goal node*). *Path planning* ini

umum digunakan dalam bidang navigasi kendaraan sekaligus juga dapat digunakan untuk mencari jalan yang tidak terhalang oleh suatu halangan (*obstacle*).

BAB IV IMPLEMENTASI DAN PENGUJIAN

4.1 Pseudocode Program Utama

```
#Parent Class
class Base_Class_Searcher
  attributes:
    string filename
    string startPath
    List of string search_log
    List of string foundFilePaths
  methods:
    constructor()
    getter()
    setter()
    findContents(string path) → List of String :
      #menghasilkan isi/konten dari suatu directory
       List of string dirs_path ← Directory.GetDirectories(path)
       List of string files_path ← Directory.GetFiles(path)
       List of string contents ← dirs_path ∪ files_path
       → contents

 #Child Class BFS
 class BFS : Base_Class_Searcher
   attributes:
     boolean findAllOccurences
     Queue of string uncheckedPaths
     boolean fileIsFound
   methods:
     FilePathFound FileLocation(string path)
     constructor()
     getter()
     setter()
     #main BFS algorithm
     findFileBFS(string startPath, boolean findAllOccurrences,
                 output : List of string this.search_log,
                 this.foundFilePaths, this.uncheckedPaths)
        this.findAllOccurrences ← findAllOccurrences
        this.uncheckedPaths.enqueue(this.startPath)
```

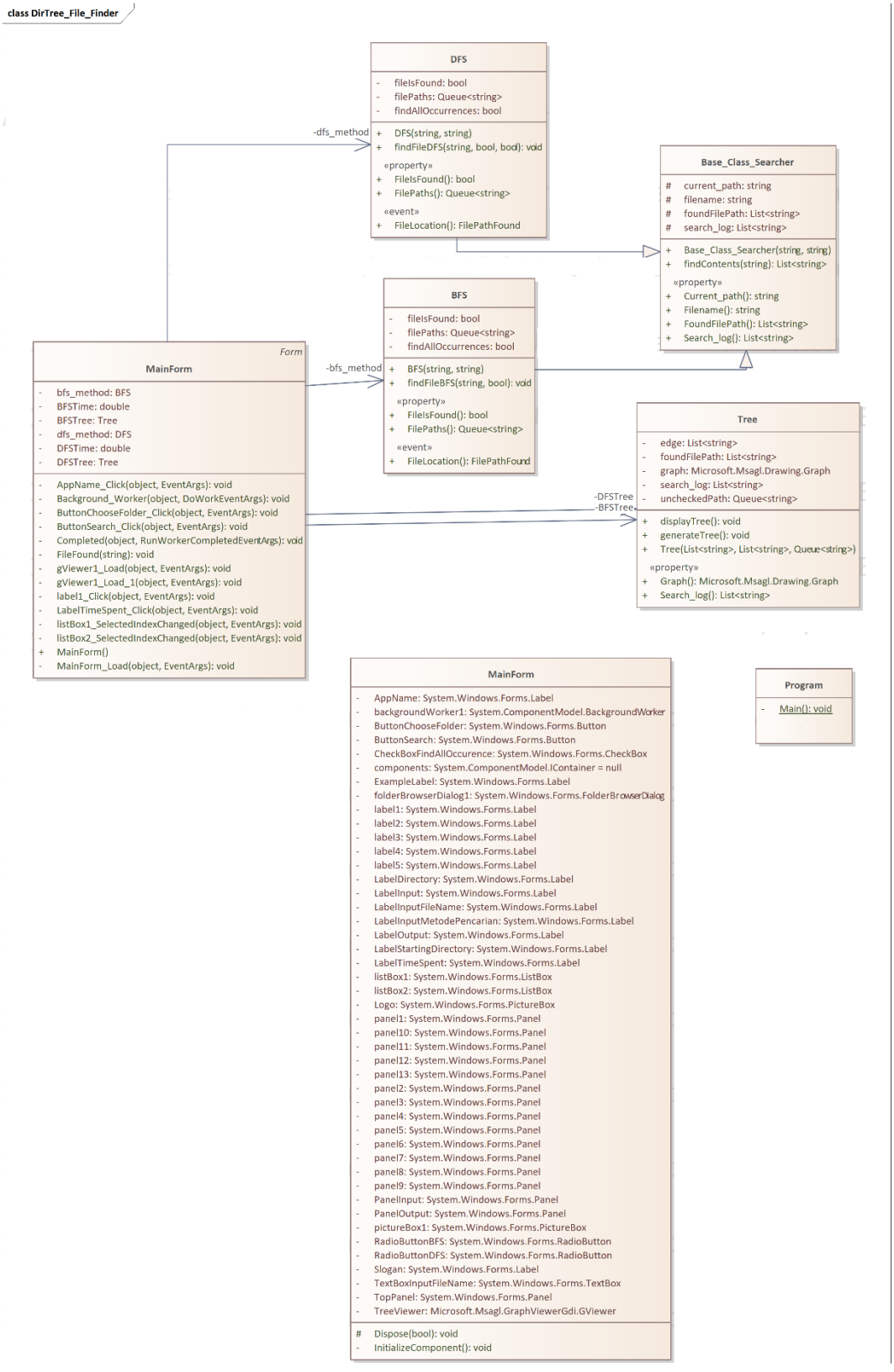
```
if (not(this.findAllOccurrences)):
    while (not(this.fileIsFound)) do:
        string path ← this.uncheckedPaths.Dequeue()
        this.search_log.Add(path)
        List of string contents ← this.findContents(path)
        #find contents secara default dir dulu, akan tetapi
         bfs file dulu maka di-reverese
        contents.Reverse()
        for (string c in contents):
            if (Path.GetFileName(c) = this.filename and
                not(this.fileIsFound)) then:
                this.search_log.Add(c)
                this.foundFilePath.Add(c)
                this.FileLocation(c)
            else if (Path.GetFileName(c) ≠ this.filename and
                      not(Directory.Exists(c)) and
                      not(this.fileIsFound)) then:
                this.search_log.Add(c)


            else if (Directory.Exists(c) or this.fileIsFound)
                      then:
                this.uncheckedPaths.Enqueue(c)
 else:
    while (this.uncheckedPaths.Length > 0) do:
       string path ← this.uncheckedPaths.Dequeue()
       this.search_log.Add(path)
       List of string contents ← this.findContents(path)
       contents.Reverse()
       for (string c in contents):
           if (Path.GetFileName(c) = this.filename and
               not(this.fileIsFound)) then:
               this.search_log.Add(c)
               this.foundFilePath.Add(c)
               this.FileLocation(c)
           else if (Path.GetFileName(c) ≠ this.filename and
                     not(Directory.Exists(c)) and
```

```
                        not(this.fileIsFound)) then:
                    this.search_log.Add(c)


                else if (Directory.Exists(c) or this.fileIsFound)
                        then:
                    this.uncheckedPaths.Enqueue(c)

#Child Class DFS
class DFS : Base_Class_Searcher
  attributes:
    boolean findAllOccurences
    Queue of string uncheckedPaths
    boolean fileIsFound
  methods:
    FilePathFound FileLocation()
    constructor()
    getter()
    setter()
    #main DFS algorithm
    findFileDFS(string startPath, boolean findAllOccurrences,
                boolean findAll, output : List of string
                this.search_log, this.foundFilePaths,
                this.uncheckedPaths):
        this.findAllOccurences ← findAllOccurrences
        List of string contents ← this.findContents(startPath)
        for (string c in contents) :
           if (this.fileIsFound and not(this.findAllOccurrences))
               then:
               this.uncheckedPaths.Enqueue(c)
           else
               this.search_log.Add(c)
               if (Directory.Exists(c) and (c ≠ '.' or c ≠ "..")
                   then:
                   this.findFileDFS(c, this.findAllOccurrences,
                                    findAll)
```

```
                else if (Path.GetFileName(c) = this.filename) then
                    this.FileLocation(c)
                    this.foundFile.Path.Add(c)
                    this.fileIsFound ← true
                    if (findAll = false) then:
                        this.findAllOccurrences = false
#Tree Generator (MSAGL)
class Tree
  attributes:
    List of string search_log
    List of string edge
    List of string foundFilePaths
    Queue of string uncheckedPaths
    MSAGL Graph graph
  methods:
    constructor()
    getter()
    setter()
    #Main tree generator algorithm (MSAGL)
    generateTree(output : MSAGL Graph this.graph):
       for (string p in this.uncheckedPaths):
           Add all edge in p as black #unexplored path
       for (string p in this.search_log):
           Add all edge in p as red   #explored path
       for (string p in this.foundFilePath)
           Add all edge in p as blue  #found path

#Main Form
class MainForm : Form
  attribute:
      double DFSTime
      double BFSTime
      DFS dfs_method
      BFS bfs_method
      Tree DFSTree
```

```
    Tree BFSTree
methods:
    constructor() #set all to null, add file found in every method
    FileFound(string path)
          listbox1.Items.Add(path) #path added into listbox1items
          listbox2.Items.Add(path_directories)
    Completed()
          if (RadioButtonBFS.Checked)
                Run(BFSTime)
          If (RadioButtonDFS.Checked)
                Run(DFSTime)
    Background_Worker()
          #program does some necessary procedure here
          Initialize stopwatch
          Initialize tree
          Create list search_log, filepath
    ButtonChooseFolder_Click()
          Create FolderBrowserDialog
          if (equals):
                LabelDirectory.Text = selectedPath
    ButtonSearch_Click()
          Clear all listbox
          Do background_worker
    listBox1_SelectedIndexChanged()
          start process if item is clicked
    listBox2_SelectedIndexChanged()
          start process if item is clicked
    #Main form algorithm
```

4.2 Struktur Data Program dan Spesifikasi Program



Lingkungan bahasa C# yang berorientasi objek membuat semua program beserta struktur data-nya dienkapsulasi menjadi suatu objek. Contohnya adalah class yang ada pada

pseudocode merupakan bagian penting dari program. Ketika dijalankan program akan menginstantiasi semua class tersebut dan akan menjalankan program dengan memanggil service yang tersedia dari objek-objek tersebut.

Terdapat beberapa objek tambahan yang merupakan *built-in* dari C# seperti objek Directory, objek Path, dan objek FilePathFound. Objek *built-in* ini membantu memudahkan operasi-operasi dasar pemrosesan file dalam program. Bagian GUI (*Graphical User Interface*) ditangani oleh objek form yang sebenarnya menginstantiasi semua objek yang bekerja di belakang (*backend*) seperti Tree, BFS, dan DFS kemudian menampung masing-masing data pada objek untuk diolah dan divisualisasi.

GUI akan menampilkan beberapa informasi:

1. Tombol untuk memilih path awal
2. Input nama file
3. Check box apakah mencari semua kemunculan atau sekali saja
4. Input metode pencarian (BFS atau DFS)
5. Directory dari file yang ditemukan
6. Absolute path dari file yang ditemukan
7. Waktu yang menghitung seberapa lama algoritma bekerja

4.3 Tata Cara Penggunaan Program

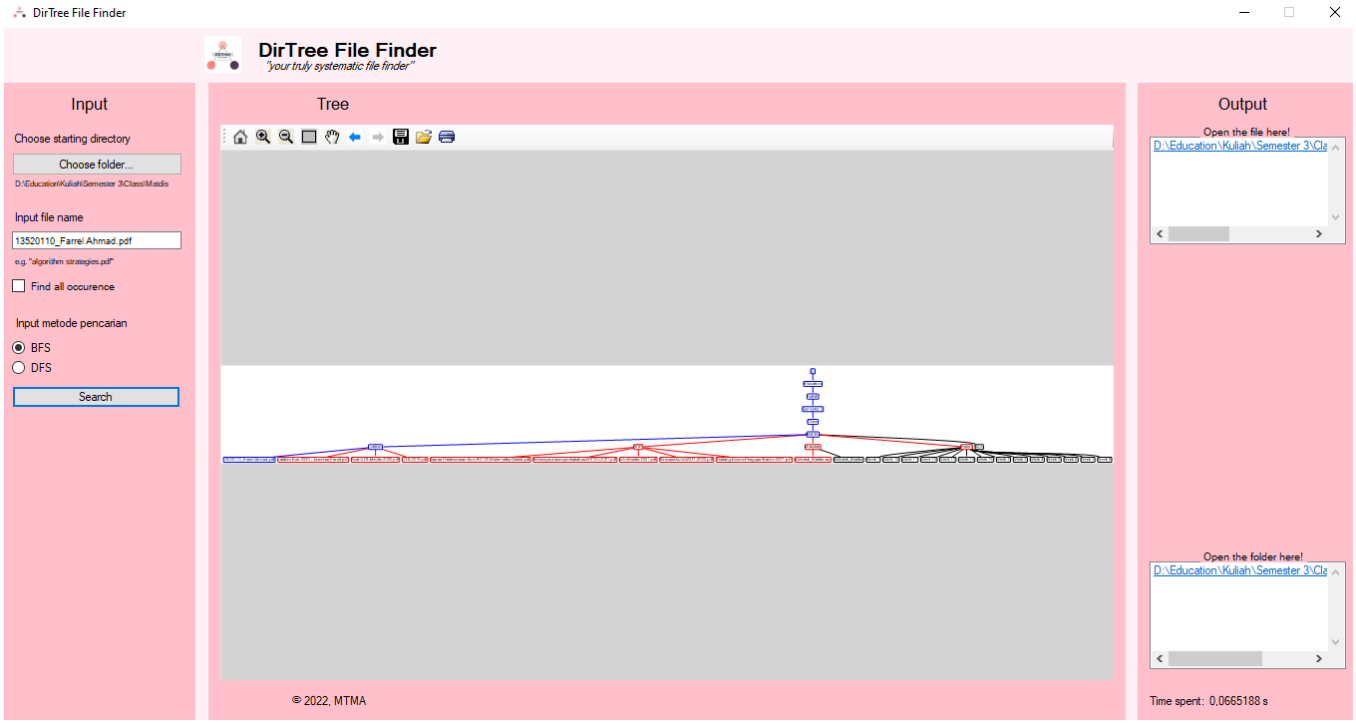
Berikut adalah langkah untuk menjalankan program:

A. Cara pertama (dengan kompilasi file):
   1. Clone repository dari link yang tertera pada subbab 4.4.
   2. Buka visual studio dan click “Open a project or a solution.”
   3. Buka directory dari hasil clone repository dan buka file Bernama `DirTree File Finder.sln`.
   4. Click start pada Visual Studio untuk compile sekaligus menjalankan program.
   5. Masukkan start path, nama file, metode pencarian, dan cari semua atau cari sekali
   6. Tekan tombol *search*.
   7. Jika ditemukan program akan langsung menampilkan diagram pohon, output hyperlink, dan waktu yang dibutuhkan untuk menemukan file tersebut.
   8. Jika tidak ditemukan akan ada pesan bahwa file tidak ditemukan.

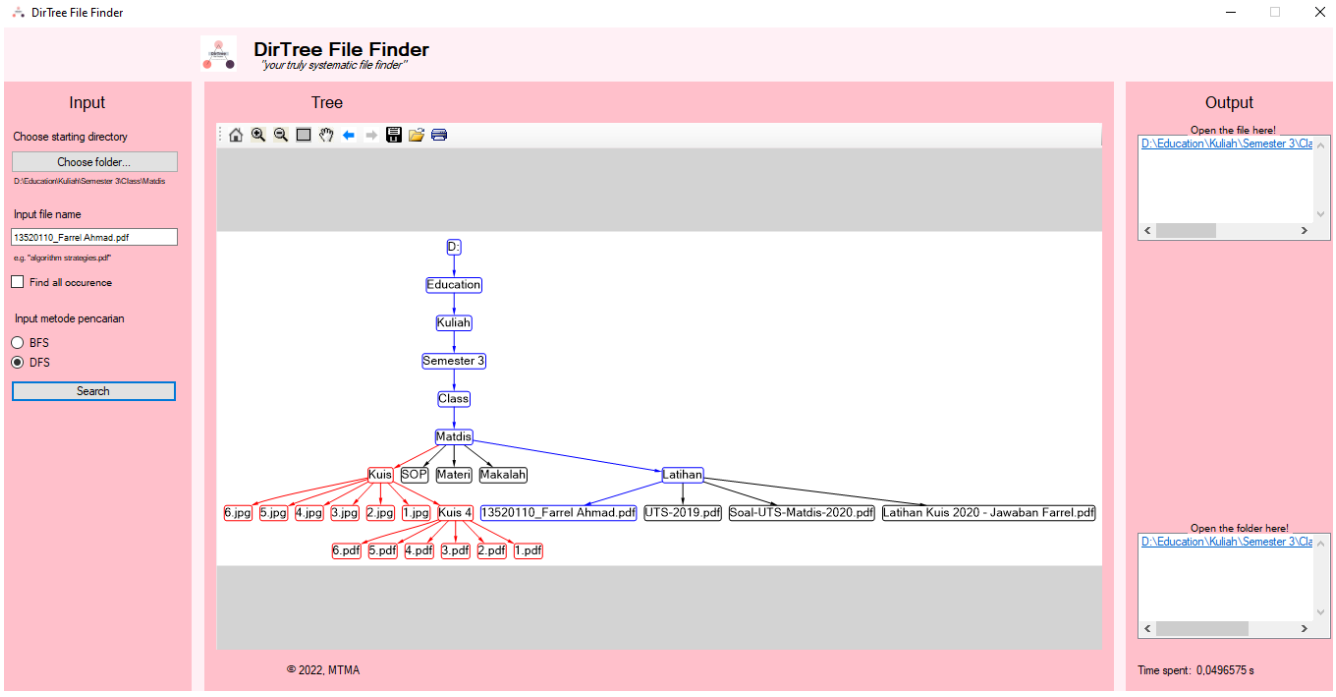
B. Cara kedua (tanpa kompilasi file):
   1. Buka directory hasil clone
   2. Buka folder bin

3. Buka `DirTree File Finder.exe`
4. Masukkan start path, nama file, metode pencarian, dan cari semua atau cari sekali
5. Tekan tombol *search*.
6. Apabila ditemukan program akan langsung menampilkan diagram pohon, output hyperlink, dan waktu yang dibutuhkan untuk menemukan file tersebut.
7. Jika tidak ditemukan akan ada pesan bahwa file tidak ditemukan.

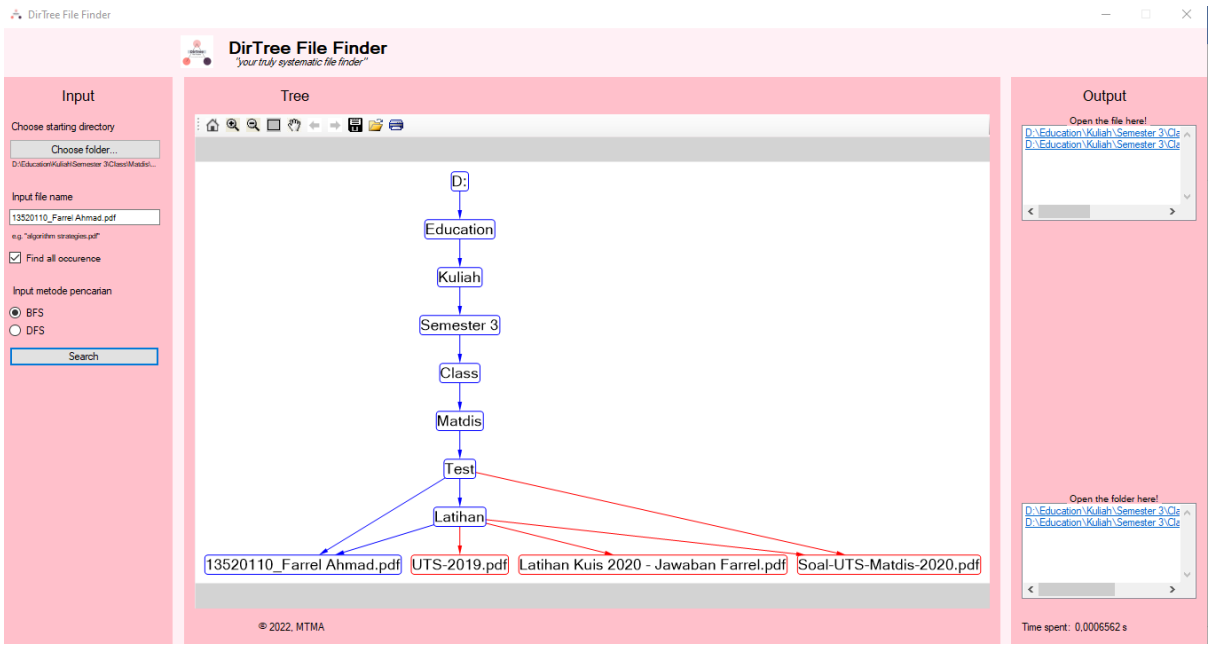
4.4 Hasil Pengujian



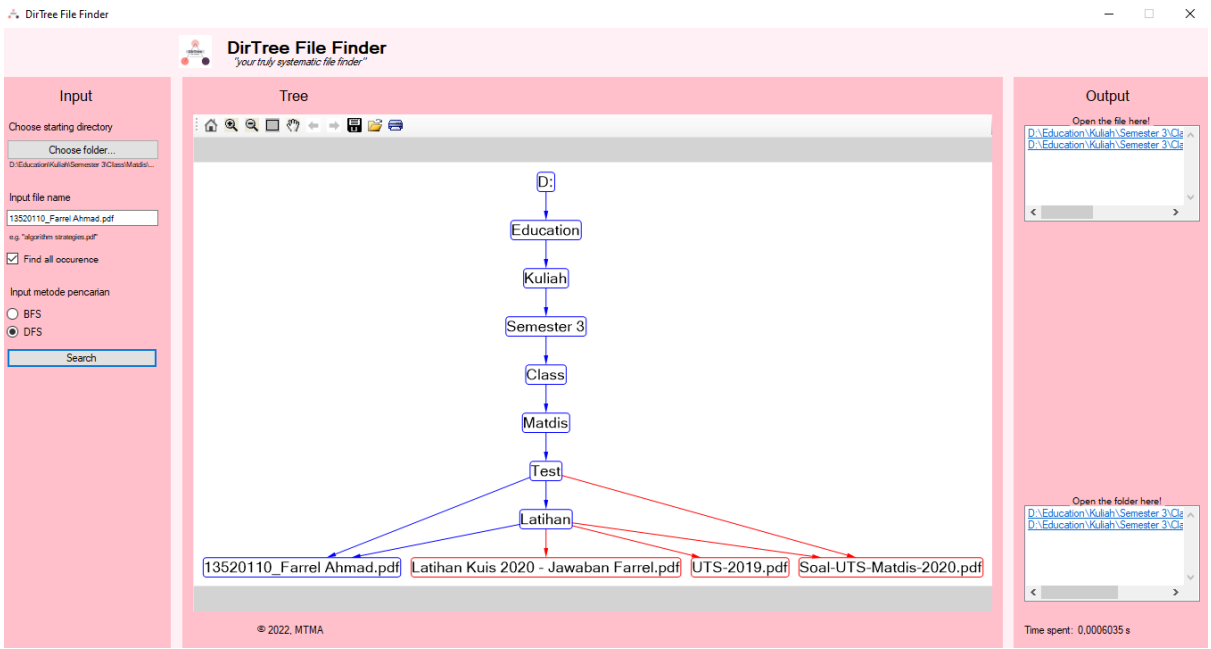
Gambar 4.4.1 Pengujian Metode Pencarian BFS tanpa *Find All Occurrences*



Gambar 4.4.2 Pengujian Metode Pencarian DFS tanpa *Find All Occurrences*



Gambar 4.4.3 Pengujian Metode Pencarian BFS dengan *Find All Occurrences*



Gambar 4.4.4 Pengujian Metode Pencarian DFS dengan *Find All Occurrences*

4.5 Analisis Desain Algoritma DFS dan BFS

Berdasarkan hasil pengujian yang telah dilakukan terhadap pencarian *file* menggunakan metode DFS dan BFS, kami mendapati bahwa masing-masing BFS maupun DFS memiliki kemungkinan lebih cepat dalam mencari *file* yang diinginkan. Dalam kasus dimana *file* berada dalam kedalaman rendah, penggunaan BFS lebih dianjurkan. Hal ini dikarenakan BFS akan secara sistematis memeriksa semua simpul di bagian atas terlebih dahulu, lalu ke simpul yang berada di bawah.

Untuk kasus di mana *file* terletak di kedalaman yang lebih tinggi, penggunaan DFS lebih dianjurkan dalam pencarian *file*. Hal ini dikarenakan DFS akan melakukan pencarian dari paling atas sampai ke kedalaman tertinggi simpulnya terlebih dahulu, lalu melakukan *backtracking* untuk mengecek simpul lainnya. Analisis kami berdasarkan pada waktu yang dibutuhkan untuk mencari *file* yang dapat dilihat pada gambar pengujian program.

Algoritma BFS memiliki kompleksitas waktu O(V + E) dengan V adalah *Vertex* (simpul) dan E adalah *Edge* (sisi). Algoritma DFS juga memiliki kompleksitas waktu O(V + E). Meskipun keduanya memiliki kompleksitas waktu yang sama, salah satu baik BFS ataupun DFS lebih optimal pada kasus-kasus tertentu. Contohnya berdasarkan hasil uji coba bahwa BFS lebih optimal untuk pencarian file yang berada di kedalaman rendah sedangkan DFS lebih optimal untuk pencarian file yang berada di kedalaman tinggi.

BAB V KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil implementasi program dapat diambil kesimpulan:

1. Algoritma BFS dan DFS untuk pencarian file berhasil diimplementasikan pada program pencari file berbasis desktop menggunakan bahasa C#.
2. Program juga berfungsi dengan baik karena dapat menemeukan file apabila memang ada dan dapat memberitahu kepada pengguna kalau file memang tidak ada ketika tidak ditemukan.
3. Algoritma BFS lebih baik daripada DFS untuk file yang memiliki kedalaman directory yang rendah.
4. Algoritma DFS lebih baik daripada BFS untuk file yang memiliki kedalam directory yang tinggi.

5.2 Saran

Berdasarkan kesimpulan tersebut dapat diambil saran berupa:

1. Perlunya optimasi heuristik yang lebih banyak pada implementasi algoritma BFS dan DFS agar dapat lebih cepat mencari file.
2. Kode pada program perlu dibuat lebih rapih lagi sehingga dibutuhkan *refactor* program yang lebih baik agar kode menjadi lebih mudah dipahami dan mudah dibaca.
3. Perlunya visualisasi graf pohon yang lebih baik agar lebih mudah terlihat untuk pencarian yang memiliki banyak *file*/*folder*.

REFERENSI

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf, diakses pada tanggal 23 Maret 2022 pukul 22.55

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf, diakses pada tanggal 23 Maret 2022 pukul 22.56

https://docs.microsoft.com/en-us/dotnet/csharp/, diakses pada tanggal 17 Maret 2022 pukul 19.15

https://www.tutorialspoint.com/difference-between-bfs-and-dfs#:~:text=Time%20Complexity%20of%20BFS%20%3D%20O,vertices%20and%20E%20is%20edges, diakses pada tanggal 25 Maret 2022 Pukul 10:46

LAMPIRAN

Link Github : https://github.com/farrel-a/Tubes2_13520047

Link Demo : https://youtu.be/pfb2sx7iP5I