TUGAS BESAR 1 IF2211 STRATEGI ALGORITMA

PENGAPLIKASIAN ALGORITMA BREADTH FIRST SEARCH (BFS) DAN DEPTH FIRST SEARCH (DFS) DALAM IMPLEMENTASI FOLDER CRAWLING

DIBUAT OLEH

13520062 - Rifqi Naufal Abdjul

13520105 - Malik Akbar Hashemi Rafsanjani

13520122 - Alifia Rahmah

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

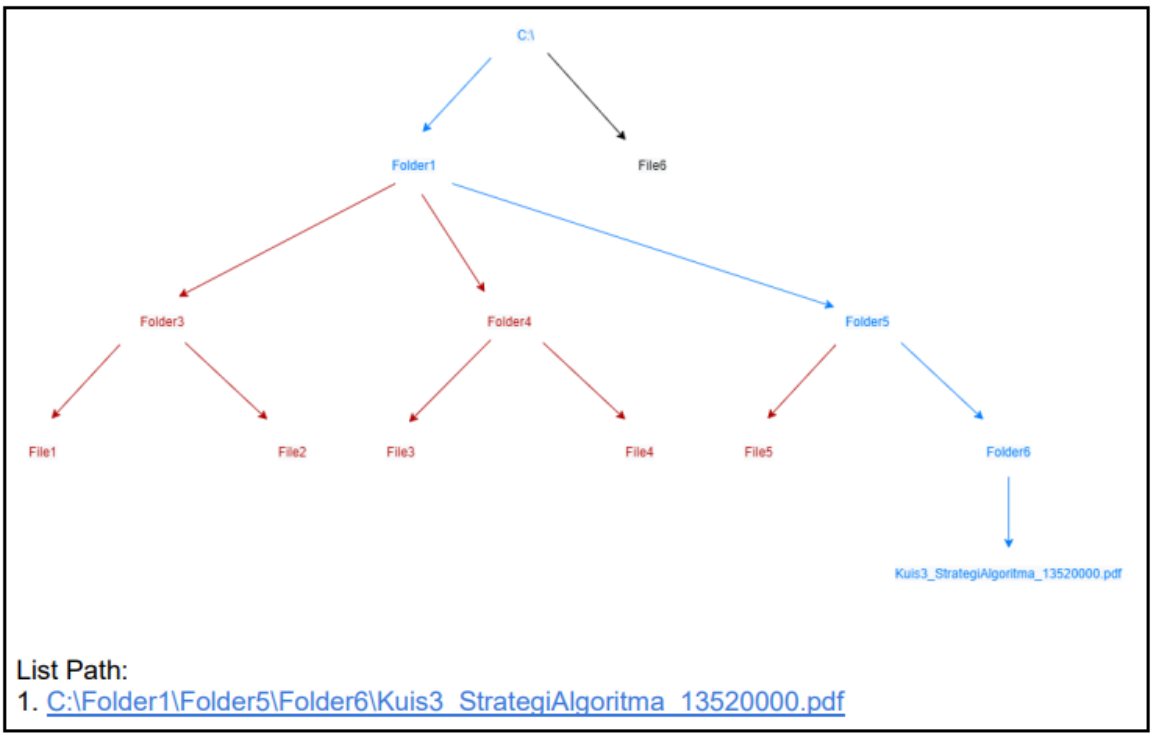
Jl. Ganesha 10, Coblong, Bandung 40132

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.

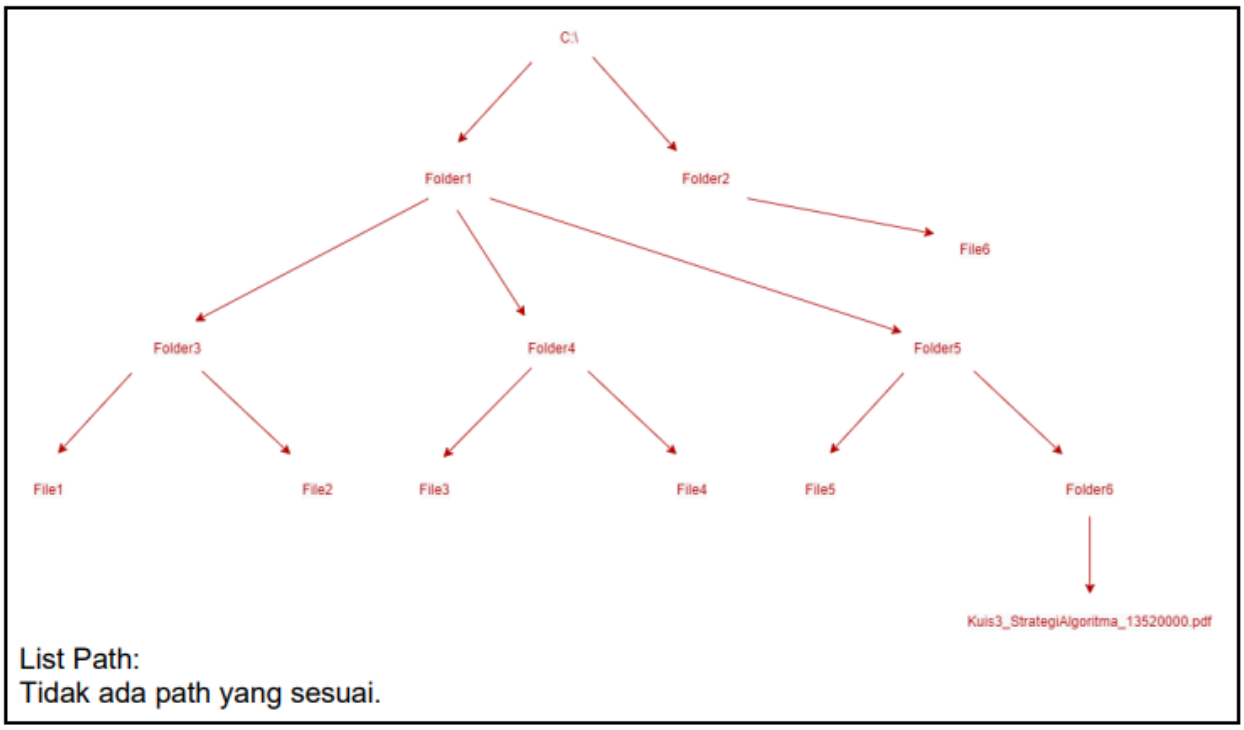
Selain pohon, Anda diminta juga menampilkan list path dari daun-daun yang bersesuaian dengan hasil pencarian. Path tersebut diharuskan memiliki hyperlink menuju folder parent dari file yang dicari, agar file langsung dapat diakses melalui browser atau file explorer. Contoh hal-hal yang dimaksud akan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



Gambar 1.1. Contoh output program jika file ditemukan

Misalnya pengguna ingin mengetahui langkah folder crawling untuk menemukan file Kuis3_StrategiAlgoritma_13520000.pdf. Maka, path pencarian DFS adalah sebagai berikut. C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf. Pada gambar di atas, rute yang dilewati pada pencarian DFS diwarnai dengan warna merah. Sedangkan, rute untuk menuju tempat file berada diberi warna biru. Rute

yang masuk antrian tapi belum diperiksa diberi warna hitam. Anda bebas menentukan warnanya asalkan dibedakan antara ketiga hal tersebut.



Gambar 1.2. 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.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Graf adalah gambar yang merepresentasikan objek-objek diskrit dan hubungannya. Dalam graf, terdapat simpul (*node*) dan sisi (*edge*). Traversal graf adalah teknik untuk mengunjungi semua simpul dalam graf secara sistematik, dengan asumsi semua graf terhubung. Jika graf adalah sebuah representasi dari persoalan, maka traversal dapat diibaratkan sebagai pencarian solusi dari persoalan tersebut. Terdapat dua tipe pengunjungan graf, yaitu pengunjungan melebar dan pengunjungan mendalam. Dalam graf statis, yaitu graf yang sudah terbentuk sebelum proses pencarian dilakukan, terdapat pendekatan traversal graf *breadth first search* dan *depth first search*.

2.1.1 Breadth First Search (BFS)

Breadth first search (BFS) adalah metode pencarian melebar. Garis besar dari pencarian ini adalah dengan mendaftarkan semua simpul yang bertetangga simpul dan mengunjunginya satu persatu hingga selesai. Langkah-langkah dari BFS adalah dengan mengunjungi salah satu simpul, lalu mengunjungi semua simpul yang bertetangga dengan simpul tersebut terlebih dahulu, lalu mengunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul yang tadi dikunjungi, demikian seterusnya hingga seluruh simpul telah dikunjungi.

2.1.2 Depth First Search (DFS)

Depth first search (DFS) adalah metode pencarian mendalam. Garis besar dari pencarian ini adalah dengan mengunjungi salah satu simpul yang bertetangga sampai semua simpul yang bertetangga sudah dikunjungi, kemudian melakukan runut balik (*backtrack*) ke simpul terakhir yang belum dikunjungi sebelumnya hingga tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

2.2 C# Desktop Application Development

2.2.1 C-Sharp (C#)

C-Sharp (C#) adalah bahasa pengembangan dari C oleh Microsoft. Bahasa C# menggunakan paradigma pemrograman berorientasi objek dan banyak digunakan dalam pembuatan aplikasi berbasis desktop dengan lingkungan .NET *framework*. Untuk mengembangkan aplikasi berbasis desktop dengan C#, biasa digunakan *integrated desktop environment* (IDE), salah satunya Microsoft Visual Studio. Untuk pengembangan graphic user interface pada C#, digunakan Windows Forms Designer yang sudah terintegrasi dengan IDE.

2.2.2 Microsoft Automatic Graph Layout (MSAGL)

Microsoft Automatic Graph Layout (MSAGL) adalah sebuah *library* .NET yang dapat digunakan untuk membuat dan menampilkan graf. Untuk menggunakan library MSAGL, diperlukan instalasi dalam NuGet *package manager*.

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Dalam *file system*, terdapat direktori (folder) dan file. Dalam direktori, bisa terdapat beberapa file atau tidak ada sama sekali. Dari folder *root* yang dipilih, dilakukan pencarian file, baik langsung dalam folder *root* atau menuju ke folder lain di dalam *root* dengan menggunakan algoritma BFS atau DFS, sesuai pilihan.

3.3.3 Langkah-Langkah Algoritma BFS

Langkah langkah untuk melakukan searching menggunakan algoritma BFS yaitu,

1. Menetapkan akar (*root*/*start directory*) sebagai titik pencarian awal
2. Melakukan pengecekan apakah titik merupakan target yang dicari
3. Jika benar, berhenti/simpan hasil
4. Jika salah, dapatkan seluruh anak dari titik tersebut, dan masukkan ke dalam *queue*
5. Dapatkan titik terdepan dalam *queue* dan gunakan titik tersebut sebagai titik pencarian
6. Ulangi dari langkah kedua hingga *queue* kosong atau target ditemukan

3.3.4 Langkah-Langkah Algoritma DFS

Langkah langkah untuk melakukan searching menggunakan algoritma DFSyaitu,

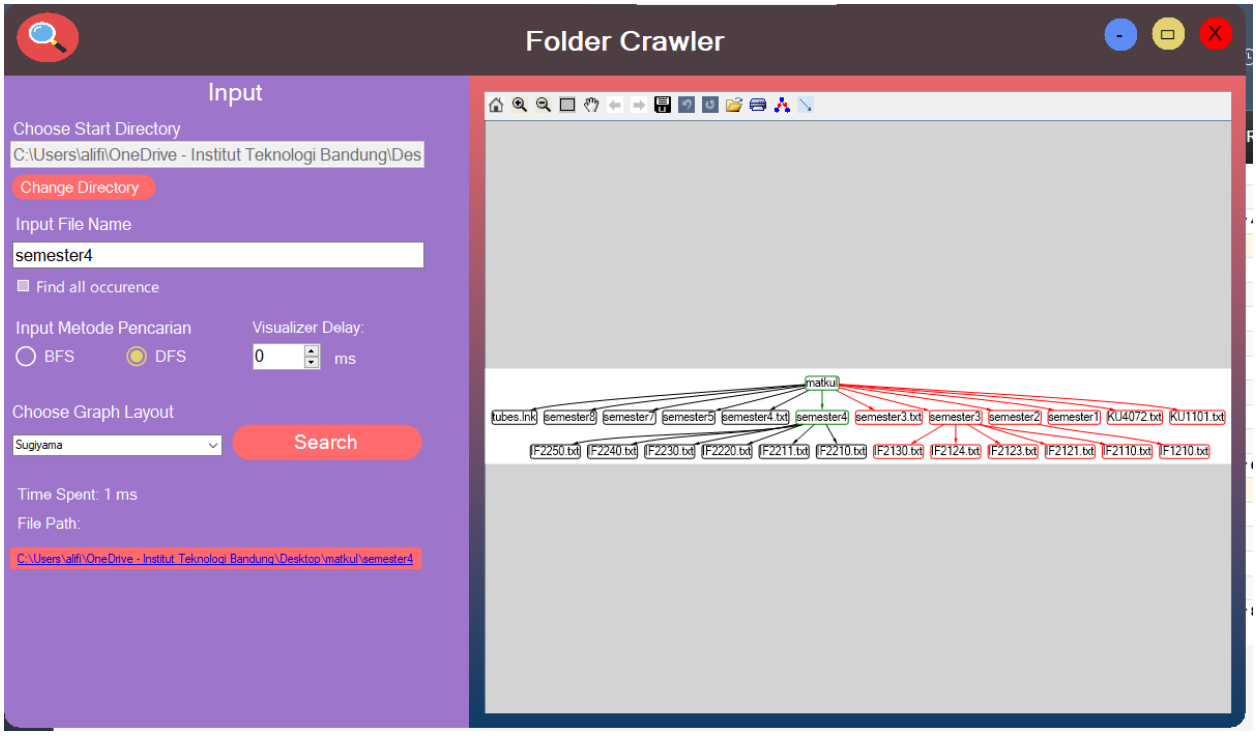
1. Menetapkan akar (*root/start directory*) sebagai titik pencarian awal
2. Melakukan pengecekan apakah titik merupakan target yang dicari
3. Jika benar, berhenti/simpan hasil
4. Jika salah, tandai titik sebagai titik yang sudah dicari dan dapatkan salah satu anak yang belum ditandai sebagai titik pencarian baru
5. Jika salah dan tidak ada anak pada titik tersebut, dapatkan induk dari titik sebagai titik pencarian baru.
6. Ulangi dari langkah kedua hingga tidak ada anak yang dapat ditelusuri pada akar

3.2 Mapping Persoalan

File system dapat digambarkan sebagai sebuah pohon (*tree*), dengan folder awal yang dipilih sebagai *root*, folder-folder di dalam folder root sebagai simpul dalam, dan file di dalam folder sebagai simpul daun dari *tree* tersebut.

3.3 Ilustrasi Kasus Lain

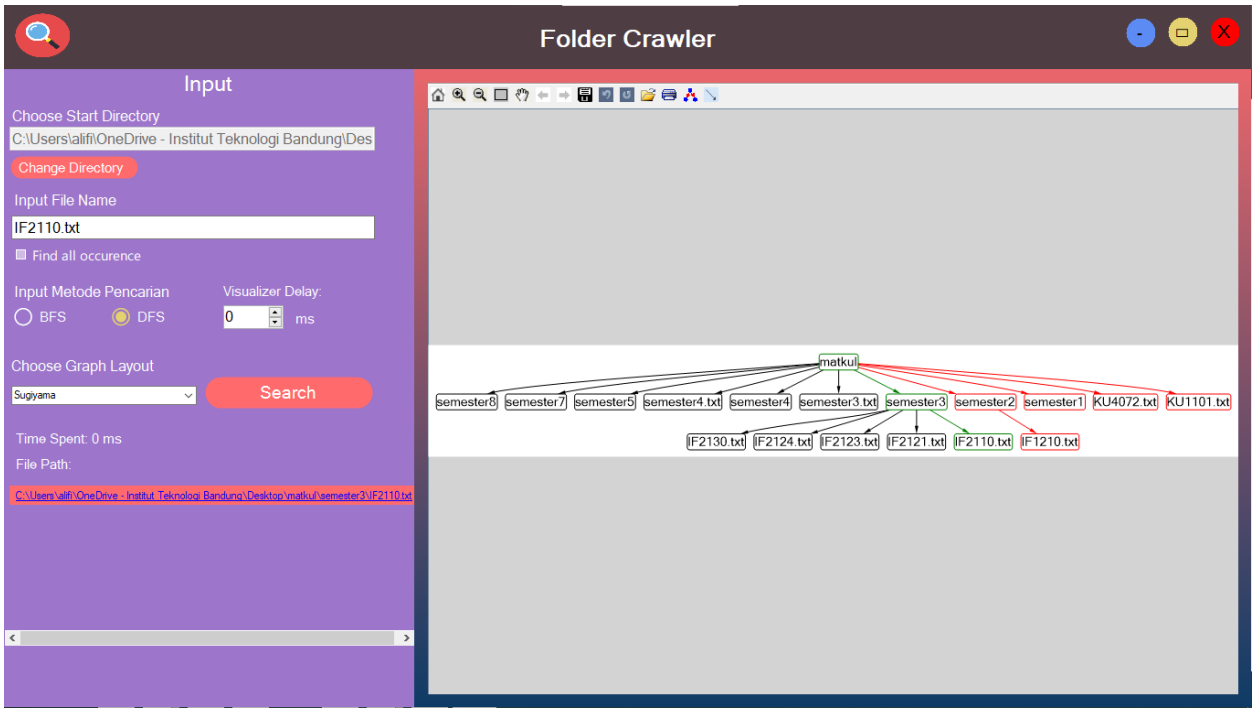
3.3.1 File dengan nama sama seperti folder



Gambar 3.3.1. Hasil pencarian “semester4” mendapatkan hasil folder semester4

Pada Gambar 3.3.1, terdapat file dengan nama semester4.txt dan folder semester4. Jika file name yang dituliskan pada input adalah “semester4” (tanpa ekstensi), program akan berhenti saat menemukan folder dengan nama semester4. File semester4.txt dan folder semester4 dibedakan dengan ekstensi (.txt), sehingga walaupun memiliki nama yang sama, tetap dapat dibedakan. Di dalam sistem operasi Windows, tidak diperbolehkan ada file dan folder dengan nama yang persis sama (file tidak memiliki ekstensi) di dalam folder yang sama, sehingga tidak akan ada file dan folder dengan nama yang persis sama dalam satu root folder.

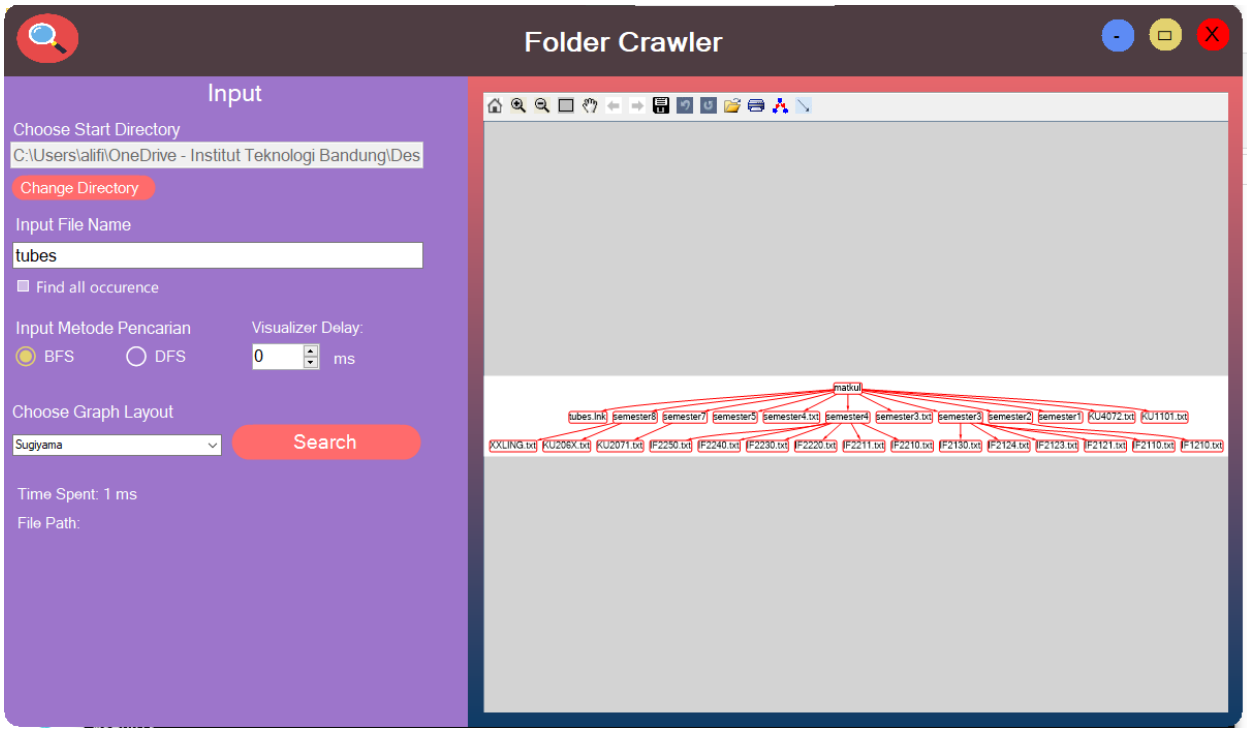
3.3.2 Folder kosong



Gambar 3.3.2. Hasil pencarian “IF2110.txt” mendapatkan hasil dalam folder semester8, dengan melewatkan folder semester5

Pada Gambar 3.3.2, terdapat folder kosong yaitu semester1. Karena tidak ada file, folder semester1 tidak akan di-expand karena tidak ada file di dalamnya. walaupun demikian, folder semester1 tetap masuk dalam Tree karena termasuk Node dalam Tree matkul.

3.3.3 Shortcut



Gambar 3.3.3. Pencarian shortcut tubes tidak menemukan apapun

Pada gambar 3.3.3, terdapat shortcut tubes, namun saat melakukan pencarian “tubes”, graf tidak menemukan apapun karena pada filesystem Windows, shortcut adalah sebuah file dengan ekstensi .lnk. Pencarian “tubes” tidak menggunakan ekstensi file sehingga hasil pencarian tidak menemukan apapun.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

4.1.1 Struktur Data (Kelas Node dan Tree)

```
class Tree
{
    public Node root;

    public Tree(string dirname)
    {
        this.root = new Node(dirname, "directory");
    }
}

class Node : IComparable<Node>
    {
        public string dirName;
        public Node parent;
        public List<Node> children;
        public string type; // "directory" or "file"

        public Node(string dirName, string type)
        {
            this.dirName = dirName;
            this.type = type;
            this.children = new List<Node>();
        }

        public Node(string dirName, string type, Node parent)
        {
            this.dirName = dirName;
            this.type = type;
            this.parent = parent;
            this.children = new List<Node>();
        }

        public int CompareTo(Node other)
        {
            return String.Compare(this.dirName, other.dirName);
        }

        public void addChildren(Node child)
        {
```

```
            if (this.type == "directory")
            {
                this.children.Append(child);
            }
        }

        public void FindChildren()
        {
            if (this.type.Equals("file"))
            {
                return;
            }
            string[] dirs = Directory.GetDirectories(this.dirName);
            string[] files = Directory.GetFiles(this.dirName);

            foreach (string dir in dirs)
            {
                this.children.Add(new Node(dir, "directory", this));
            }
            foreach (string file in files)
            {
                this.children.Add(new Node(file, "file", this));
            }

            this.children.Sort();

        }

        public bool IsAncestorOf(Node node)
        {
            return node.dirName.Contains(this.dirName);
        }

        public bool isDescendantOf(Node node)
        {
            return this.dirName.Contains(node.dirName);
        }
    }
```

4.1.2 Kelas Crawler (BFS dan DFS)

```
class Crawler
{
    public Tree tree;
    public List<(Node[] visited, Node[] looked, Node[] result)> frames;

    public Crawler(string dirname)
    {
```

```
        this.tree = new Tree(dirname);
              this.frames  =  new  List<(Node[]  visited,  Node[]  looked,  Node[]
result)>();
    }

    public void DFS(string target, bool findAll)
    {
          HashSet<Node> visited = new HashSet<Node>();     // Array of visited
nodes
          Stack<Node> looked = new Stack<Node>();           // Array of looked
nodes (in path from startDir into curNode)
         HashSet<Node> result = new HashSet<Node>();     // Array of result path
nodes
         bool flag;                                        // Flag if the node is
fully searched
        Node curNode;                                     // Current Node

        // Function to backtrack from the node
        void Backtrack()
        {
            looked.Pop();
                        this.frames.Add((visited.ToArray(),  looked.ToArray(),
result.ToArray()));
        }

        // Function to move into node
        void MoveTo(Node node)
        {
            visited.Add(node);
            looked.Push(node);
                        this.frames.Add((visited.ToArray(),  looked.ToArray(),
result.ToArray()));
        }

        // Initialize Search
        flag = false;
        curNode = this.tree.root;
        MoveTo(this.tree.root);

        while (!flag || curNode.parent is object)
        {
            if (curNode.children.Count == 0)
            {
                curNode.FindChildren();
            }
            // Target is found
            if (Utils.GetDirName(curNode.dirName).Equals(target))
            {
                foreach (Node node in looked)
```

```
                {
                    result.Add(node);
                    Console.WriteLine(result.Count);
                }
                if (!findAll)
                {
                    break;
                }
            }


            // Flag is true if there are no children to check
            flag = true;
            if (curNode.children.Count > 0)
            {
                foreach (Node child in curNode.children)
                {
                    if (!visited.Contains(child))
                    {
                        flag = false;
                        break;
                    }
                }
            }

                // Backtrack if there's no children to check and curNode have
parent
            if (flag && curNode.parent is object)
            {
                curNode = curNode.parent;
                Backtrack();
                flag = false;
            } else
            // Go to unvisited children;
            {
                foreach (Node child in curNode.children)
                {
                    if (!visited.Contains(child))
                    {
                        curNode = child;
                        MoveTo(child);
                        break;
                    }
                }
            }
        }

        // Clear looked stack when finished
        looked.Clear();
```

```
                        this.frames.Add((visited.ToArray(),   looked.ToArray(),
result.ToArray()));
    }

    public void BFS(string target, bool findAll)
    {
          HashSet<Node> visited = new HashSet<Node>();     // Array of visited
nodes
          Stack<Node> looked = new Stack<Node>();           // Array of looked
nodes (in path from startDir into curNode)
         HashSet<Node> result = new HashSet<Node>();     // Array of result path
nodes
        Queue<Node> queue = new Queue<Node>();          // Search Queue
        Node curNode;                                   // Current Node

        // Function to backtrack from the node
        void Backtrack()
        {
            looked.Pop();
        }

        // Function to move into node
        void MoveTo(Node node)
        {
            visited.Add(node);
            looked.Push(node);
                        this.frames.Add((visited.ToArray(),  looked.ToArray(),
result.ToArray()));
        }

        // Initialize Search
        queue.Enqueue(this.tree.root);

        do
        {

            if (!this.tree.root.Equals(queue.Peek()))
            {
                // Backtrack until it's ancestor of next node
                while (!looked.Peek().IsAncestorOf(queue.Peek()))
                {
                    Backtrack();
                }

                // MoveTo until it's the parent of next node
                if (queue.Peek().parent is object)
                {
                            while (looked.Peek().IsAncestorOf(queue.Peek()) &&
!queue.Peek().parent.Equals(looked.Peek()))
```

```
                    {

                        foreach (Node child in looked.Peek().children)
                        {
                            if (child.IsAncestorOf(queue.Peek()))
                            {
                                MoveTo(child);
                                break;
                            }
                        }
                    }
                }
            }

            // Get Next Node
            curNode = queue.Dequeue();

            // Move to next Node
            MoveTo(curNode);

            // Get the children of the node
            curNode.FindChildren();

            foreach (Node child in curNode.children)
            {
                // Move to the child
                MoveTo(child);
                // Terminates if the child is the target
                if (Utils.GetDirName(child.dirName).Equals(target))
                {
                    foreach (Node node in looked)
                    {
                        result.Add(node);
                        Console.WriteLine(result.Count);
                    }
                    if (!findAll)
                    {
                        break;
                    } else
                    {
                        Backtrack();
                    }

                } else
                    // Else, add the directories into queue, then backtrack to
current node
                {
                    if (child.type.Equals("directory"))
                    {
```

```
                        queue.Enqueue(child);
                    }
                    Backtrack();
                }
            }
                } while (!Utils.GetDirName(curNode.dirName).Equals(target) &&
queue.Count > 0);

        // Clear looked stack when finished
        looked.Clear();
                        this.frames.Add((visited.ToArray(), looked.ToArray(),
result.ToArray()));
    }

    public string[] GetResults(string target)
    {
        List<string> res = new List<string>();
        foreach (Node node in frames.Last().result)
        {
            if (Utils.GetDirName(node.dirName).Equals(target))
            {
                res.Add(node.dirName);
            }
        }
        return res.ToArray();
    }
```

4.1.3 Kelas Visualizer

```
class Visualizer
{
    private DirTree.Tree tree;

      private (DirTree.Node[] visited, DirTree.Node[] looked, DirTree.Node[]
result)[] frames;
          public Visualizer(DirTree.Tree tree, (DirTree.Node[] visited,
DirTree.Node[] looked, DirTree.Node[] result)[] frames)
    {
        this.tree = tree;
        this.frames = frames;
    }

    public Visualizer(Crawler crawler)
    {
        this.tree = crawler.tree;
        this.frames = crawler.frames.ToArray();
    }
```

```
    public async void Visualize(Panel target, int delay)
    {
        GViewer viewer = new GViewer();
        viewer.Dock = DockStyle.Fill;
        target.SuspendLayout();
        target.Controls.Clear();
        target.Controls.Add(viewer);
        target.ResumeLayout();
         foreach ((DirTree.Node[] visited, DirTree.Node[] looked, DirTree.Node[]
result) frame in this.frames)
        {
            Graph temp = VisualizeTree(this.tree, frame);
            double tempZoomF = viewer.ZoomF;
            viewer.Graph = temp;
            viewer.ZoomF = tempZoomF;
            await Task.Delay(delay);
        }
    }

      public Graph VisualizeTree(DirTree.Tree tree, (DirTree.Node[] visited,
DirTree.Node[] looked, DirTree.Node[] result) frame)
    {
        Graph graph = new Graph();
        Node node = graph.AddNode(tree.root.dirName);
        node.LabelText = Utils.GetDirName(tree.root.dirName);
        if (frame.looked.Contains(tree.root))
        {
            node.Attr.Color = Color.Blue;
        }
        else if (frame.result.Contains(tree.root))
        {
            node.Attr.Color = Color.Green;
        }
        else if (frame.visited.Contains(tree.root))
        {
            node.Attr.Color = Color.Red;
        }
        VisualizeChildren(graph, tree.root, frame);
        return graph;
    }

        private void VisualizeChildren(Graph graph, DirTree.Node parent,
(DirTree.Node[] visited, DirTree.Node[] looked, DirTree.Node[] result) frame)
    {
        if (parent.children.Count > 0)
        {
            foreach (DirTree.Node child in parent.children)
            {
                Node node = graph.AddNode(child.dirName);
```

```
                node.LabelText = Utils.GetDirName(child.dirName);
                Edge edge = graph.AddEdge(parent.dirName, child.dirName);
                if (frame.looked.Contains(child))
                {
                    node.Attr.Color = Color.Blue;
                    edge.Attr.Color = Color.Blue;
                } else if (frame.result.Contains(child))
                {
                    node.Attr.Color = Color.Green;
                    edge.Attr.Color = Color.Green;
                }
                else if (frame.visited.Contains(child))
                {
                    node.Attr.Color = Color.Red;
                    edge.Attr.Color = Color.Red;
                } else
                {
                    continue;
                }
                VisualizeChildren(graph, child, frame);
            }
        }
    }
}
```

4.2 Struktur Data

Struktur folder project pada tugas kali ini adalah sebagai berikut,

```
├──bin
├──doc
└──src
    ├──assets
    ├──Classes
    │   ├──Crawler
    │   ├──DirTree
    │   └──Forms
    │       └──Components
    ├──packages
    │   ├──Microsoft.Msagl.1.1.3
    │   │   └──lib
    │   │       └──net40
    │   ├──Microsoft.Msagl.Drawing.1.1.3
    │   │   └──lib
    │   │       └──net40
    │   └──Microsoft.Msagl.GraphViewerGDI.1.1.3
    │       └──lib
    │           └──net40
    │
```

```
└──Properties
```

4.3 Tata Cara Penggunaan Program

Untuk melakukan setup program ini, dibutuhkan beberapa *dependencies* seperti,

1. .NET Framework minimal 4.7.2
2. MSBuild minimal 16.0 (Dapat didapatkan saat meng-*install* Visual Studio 2019 ke atas)

Berikut langkah-langkah untuk melakukan setup program

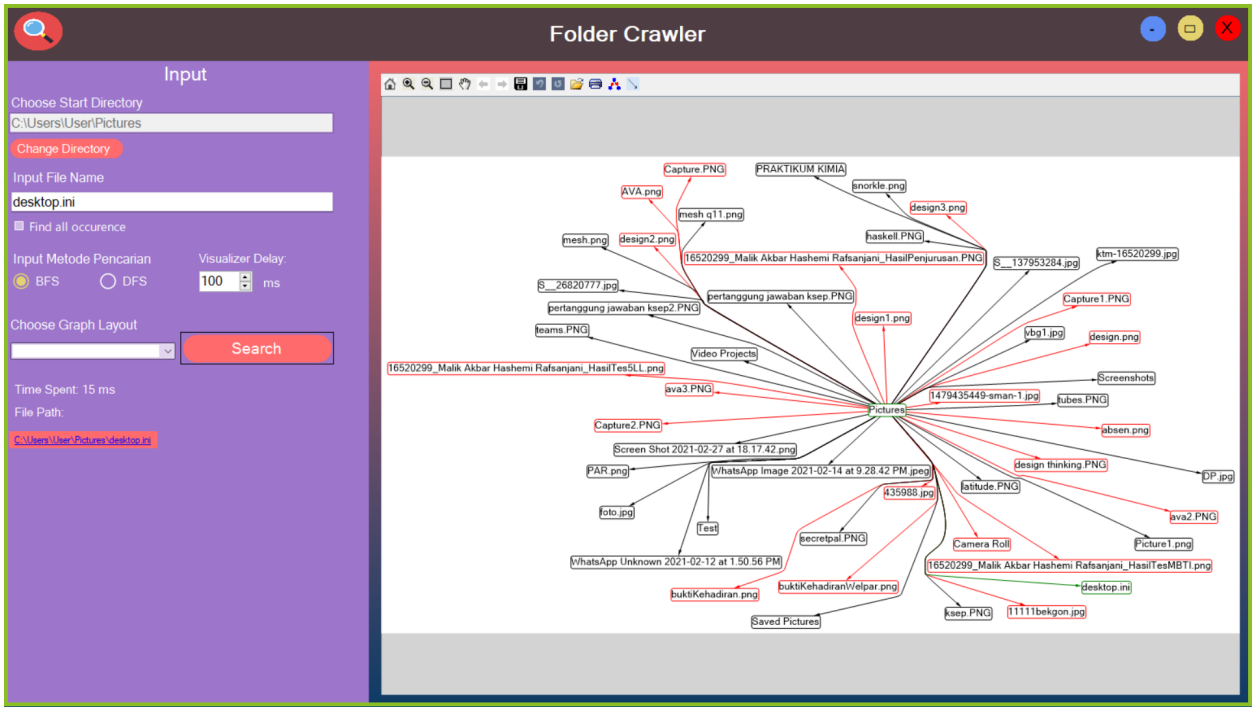
1. Buka command line pada direktori */src*
2. Jalankan command *msbuild* pada command line
3. Jalankan program FolderCrawler.exe pada folder */bin*

Berikut tata cara penggunan program:

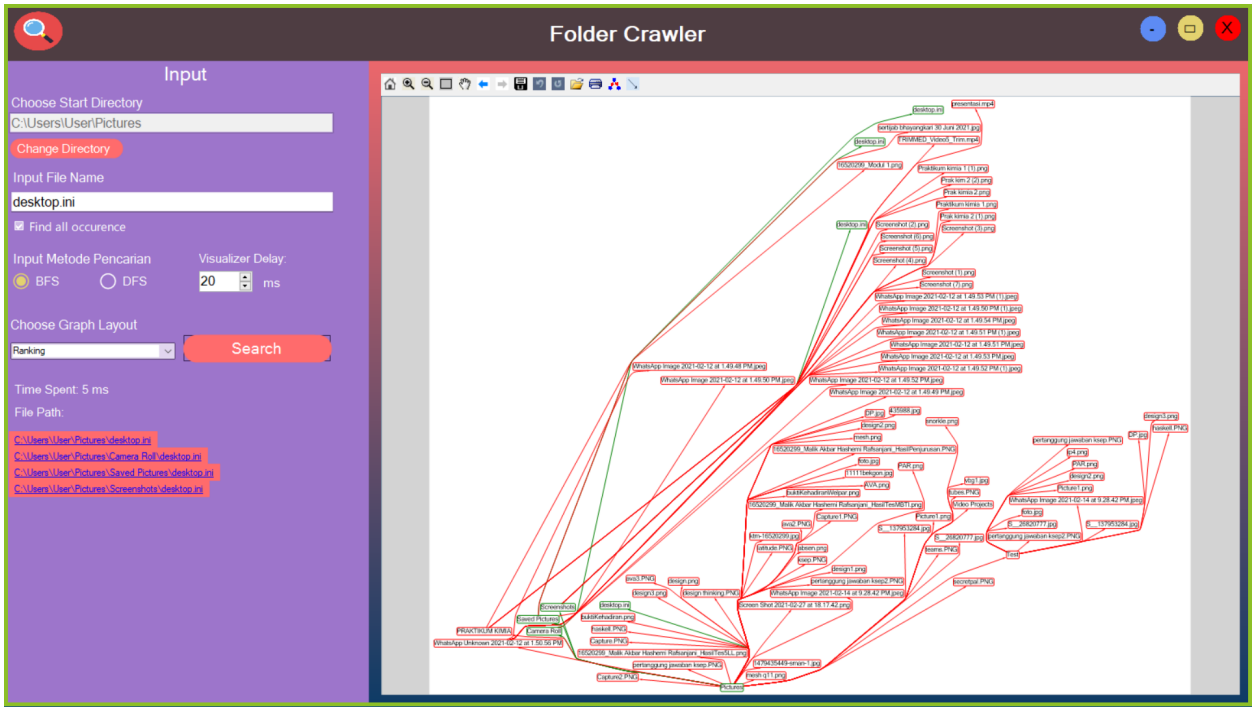
1. Buka aplikasi Folder Crawler
2. Klik tombol “Change Directory” untuk memilih folder yang akan menjadi *root*
3. Tulis nama file yang ingin dicari di input dengan label “Input File Name”
4. Pilih metode pencarian dengan memilih radio button BFS untuk melakukan pencarian *breadth first search*, atau memilih radio button DFS untuk melakukan pencarian *depth first search*
5. Pilih delay visualizer untuk menambahkan delay di tiap langkah pencarian
6. Pilih layout graf untuk visualizer
7. Lakukan pencarian dengan klik tombol “Search”
8. Pohon pencarian, waktu melakukan pencarian, dan *link file path* hasil pencarian juga dihasilkan

4.4 Hasil Pengujian

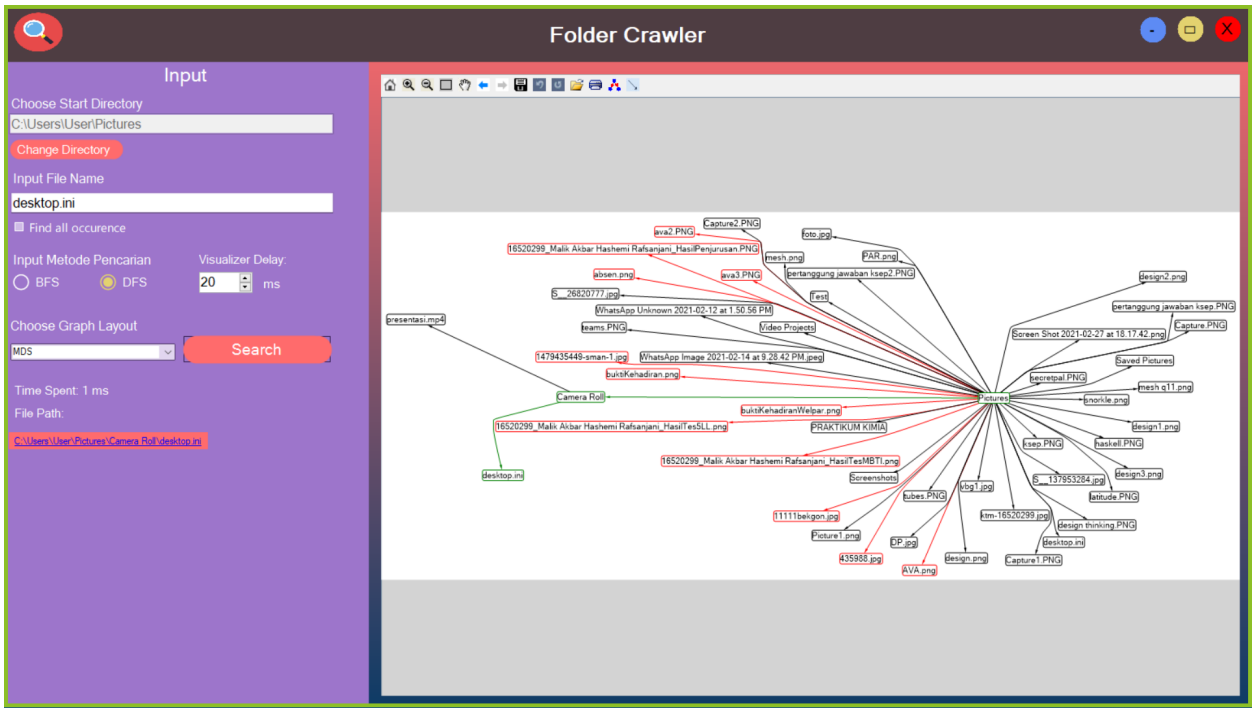
4.4.1 BFS - find one occurence



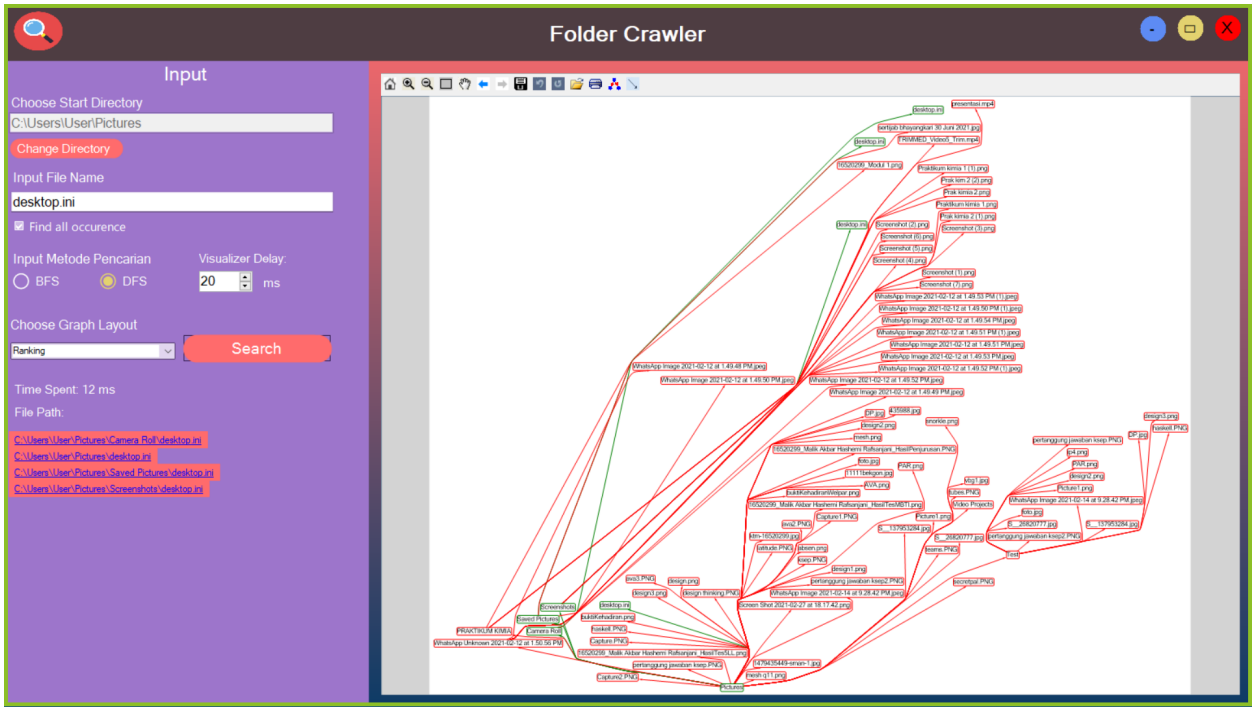
4.4.2 BFS - find all occurence



4.4.3 DFS - find one occurence



4.4.4 DFS - find all occurence



4.5 Analisis Desain Solusi

Pada tugas kali ini, terdapat dua cara solusi untuk menyelesaikan permasalahan pencarian berkas, yaitu dengan algoritma BFS dan algoritma DFS. Pada algoritma BFS, program akan menyelesaikan pencarian untuk masing-masing kedalaman sampai selesai. Pada hasil tugas kami, dapat dilihat pada hasil pengujian 4.4.1, progam berhenti mencari pada kedalaman satu karena file yang ingin dicari telah ditemukan pada kedalaman satu. Pada algoritma DFS, sistem akan menyelesaikan pencarian untuk setiap simpul dalam sampai menemui semua simpul daun kecuali menemukan simpul daun yang dicari. Pada hasil tugas kami, dapat dilihat pada hasil pengujian 4.4.3, progam berhenti mencari ketika menemukan berkas yang dicari pertama kali, pada kedalaman dua, sedangkan simpul-simpul pada kedalaman satu belum sepenuhnya selesai dicari. Untuk permasalahan pencarian semua kemunculan berkas, hasil dari algoritma BFS dengan algoritma DFS sangatlah mirip hanya berbeda di urutan hasil pencarian. Hasil dari algoritma BFS tersebut selalu urut berdasarkan kedalaman, sedangkan algoritma DFS tidak selalu terurut.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas ini, digunakan algoritma BFS (Breadth First Search) dan DFS (Depth First Search) untuk menelusuri *file system* dan mencari berkas yang diinginkan. Algoritma tersebut terbukti cukup efektif untuk menyelesaikan permasalahan ini. Hal tersebut dapat terlihat pada hasil pengujian, kedua algoritma tersebut sukses untuk menemukan berkas yang diinginkan. Selain itu, program yang telah dibuat dibungkus pada aplikasi GUI sederhana untuk memudahkan interaksi antara program dan pengguna. Aplikasi GUI ini juga dapat menampilkan visualisasi hasil pencarian dengan cukup baik. Hal tersebut juga dapat dilihat pada hasil pengujian, pemodelan pohon dari *file system* dapat dipahami dengan cukup jelas dengan disertai pula keterangan warna yang menyatakan jalur yang menghasilkan solusi, jalur yang tidak menghasilkan solusi, dan jalur yang belum diperiksa.

5.2 Saran

Dalam peng-implementasi-an program “Folder Crawler” pada tugas ini, kami menghadapi beberapa permasalahan yang menghambat proses pengerjaan kami. Selain itu, terdapat beberapa hal yang dapat diimplementasi tetapi kami memilih untuk tidak dikarenakan beberapa alasan. Saran kami dalam pengerjaan tugas ini adalah sebagai berikut,

- Melakukan naming conventions yang baik dan benar
- Desain algoritma yang jelas dan runtut sehingga sesuai dengan konvensi DRY (Don’t Repeat Yourself)
- Menjelaskan seluruh bagian kode dengan *comment* agar mudah dimengerti
- Enkapsulasi kelas yang sesuai dengan fungsi agar meningkatkan kejelasan bagian program

DAFTAR PUSTAKA

Breadth/Depth First Search, Bahan Kuliah IF2211 Strategi Algoritma. (2022).

A tour of C# - Overview. (2022). https://docs.microsoft.com/id-id/dotnet/csharp/tour-of-csharp/ (diakses tanggal 20 Maret 2022)