LAPORAN TUGAS BESAR
IMPLEMENTASI *FOLDER CRAWLING*

IF2211/Strategi Algoritma

Dipersiapkan oleh:

Pencerahan Explorer

1. Christine Hutabarat - 13520005
2. Nayotama Pradipta - 13520089
3. Muhammad Rakha Athaya - 13520108

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

BAB I
Pendahuluan

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 saja harus 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 pastinya akan sangat memakan waktu dan energi.

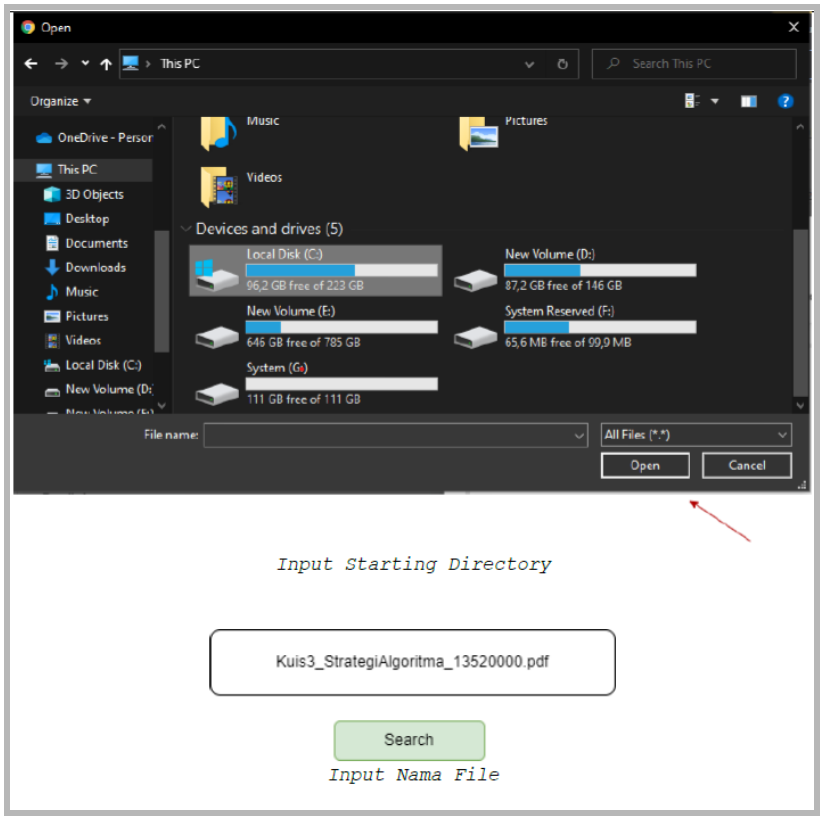
Meskipun demikian, kita tidak perlu cemas dalam menghadapi persoalan tersebut sekarang. Pasalnya, hampir seluruh sistem operasi sudah menyediakan fitur search yang dapat digunakan untuk mencari file yang kita inginkan. Kita cukup memasukkan query atau kata kunci pada kotak pencarian, dan komputer akan 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.

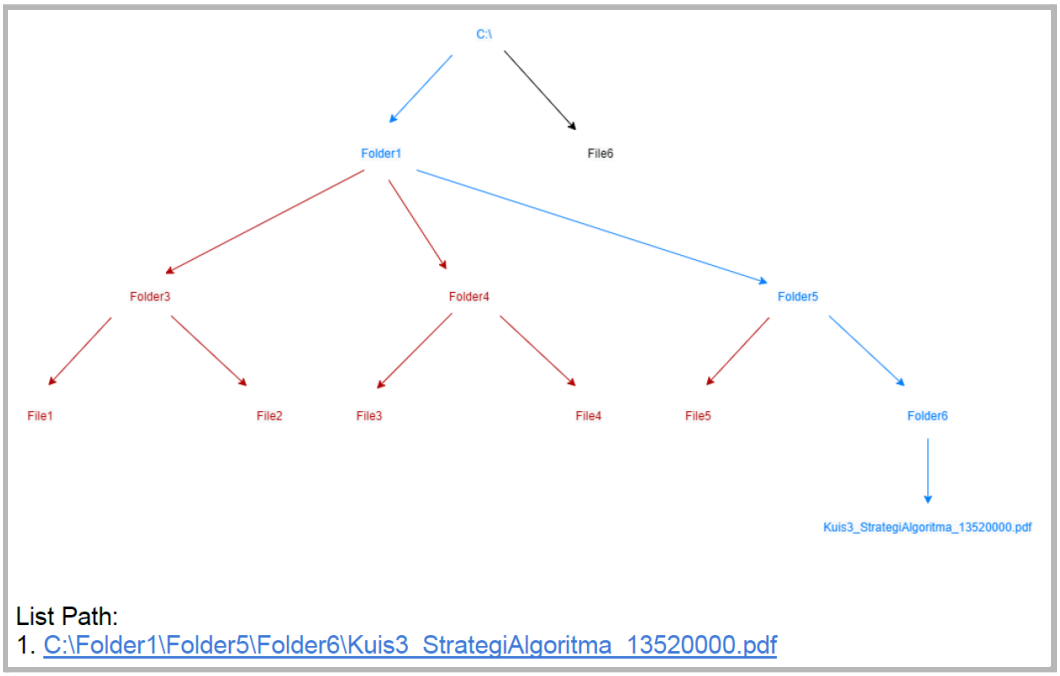
Deskripsi Tugas

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 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.

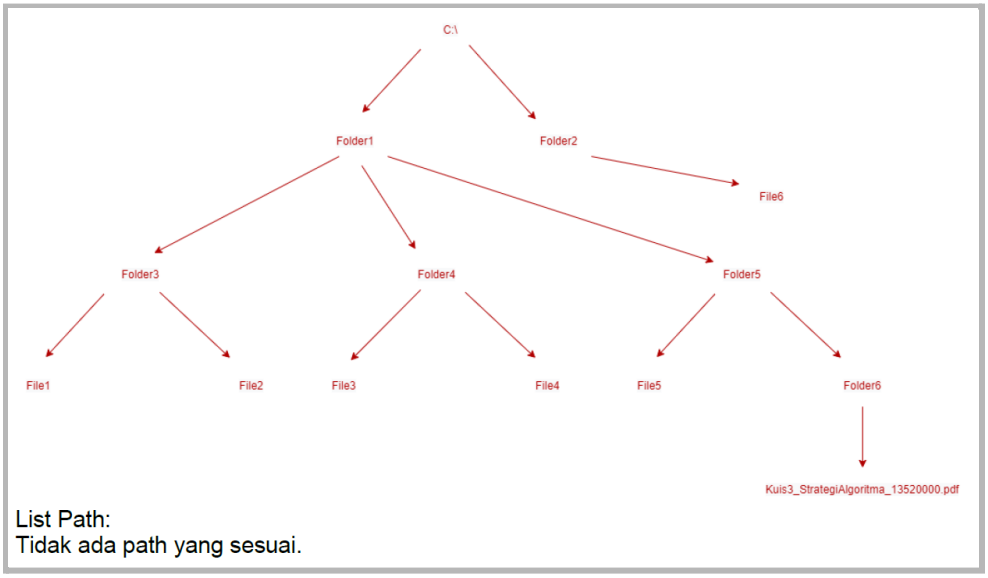


Gambar 1.1 Contoh Input Program



Gambar 1.2 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 1.3 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

Dasar Teori

Graph Traversal

Graf adalah suatu struktur yang berisi himpunan dari simpul dan himpunan dari sisi yang menghubungkan sepasang simpul. Simpul-simpul pada graf umumnya digambarkan menggunakan lingkaran sementara sisi-sisi dari graf digambarkan sebagai garis yang kedua ujungnya terhubung pada simpul. Terdapat dua jenis graf, yaitu graf berarah dan graf tidak berarah. Pada graf berarah, setiap sisi memiliki orientasi arah, dan digambarkan menggunakan panah. Graf dapat digunakan sebagai representasi dari persoalan, di mana pencarian solusi dari masalah yang diberikan dilakukan dengan menelusuri/melakukan traversal pada graf. Terdapat beberapa metode untuk melakukan traversal pada graf, yaitu metode pencarian tanpa informasi seperti pencarian melebar (BFS) dan pencarian mendalam (DFS), serta metode pencarian dengan informasi seperti metode *Best-First Search*.

Breadth-First Search

Breadth-First Search (BFS) adalah salah satu metode traversal graf yang mengutamakan pencarian secara melebar ke seluruh tetangga dari simpul yang ditinjau sebelum akhirnya meninjau dari simpul tetangga untuk melihat simpul-simpul tetangga lainnya. Pencarian menggunakan metode ini cukup efektif apabila simpul tujuan tidak berada jauh dari simpul awal (simpul awal dan simpul tujuan tidak diperantarai banyak simpul). Metode BFS memanfaatkan struktur antrian untuk menentukan simpul yang harus ditinjau berikutnya. Misalkan akan dilakukan penelusuran pada sebuah graf, dimulai dari simpul A. Penelusuran menggunakan metode BFS dimulai dari memasukkan simpul A ke dalam antrian dan ketika ditinjau dan simpul A yang merupakan kepala antrian dihapus. Berikutnya, masukkan simpul-simpul tetangga dari A ke dalam antrian, dan tandai simpul A sebagai simpul yang sudah dikunjungi. Hapus simpul yang terdapat pada kepala antrian, lalu tinjau tetangganya dan masukkan ke dalam antrian. Langkah-langkah tersebut dilakukan hingga tidak tersisa lagi simpul pada antrian.

Depth-First Search

Depth-First Search (DFS) adalah metode traversal graf yang mengutamakan penelusuran secara mendalam ke simpul tetangga. Setelah penelusuran berhenti pada simpul daun, dilakukan runut balik ke simpul-simpul sebelumnya, untuk mencari lagi simpul lain yang belum dikunjungi. Pencarian menggunakan *depth-first search* baik digunakan dalam kasus pencarian di mana simpul tujuan berada cukup jauh dari simpul awal. Pada metode ini, digunakan struktur *stack*

atau tumpukan untuk melacak dan menentukan urutan simpul yang akan ditinjau berikutnya. Misalkan akan dilakukan pencarian pada sebuah graf dengan menggunakan metode DFS, dimulai dari simpul A. Masukkan simpul A ke dalam tumpukan, kemudian hapus dan tinjau simpul A sebagai puncak dari tumpukan. Masukkan setiap tetangga dari simpul A ke dalam tumpukan, lalu tandai simpul A sebagai simpul yang telah dikunjungi. Berikutnya tinjau simpul yang berada di puncak tumpukan dan masukkan tetangga-tetangganya yang belum dikunjungi ke dalam tumpukan. Langkah-langkah tersebut dilakukan hingga ditemukan simpul tujuan atau hingga tidak tersisa lagi simpul pada tumpukan.

C# Desktop Application Development

Bahasa pemrograman C# adalah salah satu bahasa pemrograman berorientasi objek yang dikembangkan oleh Microsoft. Bahasa ini dibangun pada kerangka kerja .NET yang juga dikembangkan oleh perusahaan yang sama. Kerangka kerja .NET merupakan kerangka kerja yang dapat digunakan untuk pembangunan dan pengembangan aplikasi berbasis Windows. Umumnya pembangunan perangkat lunak dengan kerangka kerja .NET dilakukan melalui perangkat lunak IDE dan juga teks editor Visual Studio.

Bab III
Analisis Pemecahan Masalah

Langkah Pemecahan Masalah

Permasalahan *folder crawling* atau penelusuran folder dapat direpresentasikan dengan menggunakan pohon berakar, yaitu graf berarah yang tidak membentuk sirkuit dan dimulai dari sebuah simpul yang disebut sebagai akar. Setiap simpul pada pohon mewakilkan folder atau file, dan setiap anak simpul adalah folder atau file lainnya yang berada dalam folder orangtua. Dengan begitu, setiap simpul daun yang terdapat pada pohon merepresentasikan file atau folder kosong. Jika setiap simpul dinamai dengan nama folder atau file, maka susunan dari simpul orangtua ke simpul-simpul keturunannya menggambarkan *path* atau jalur penelusuran dari simpul akar ke simpul tujuan.

Pemetaan Persoalan ke Elemen BFS dan DFS

Setelah struktur dan susunan folder serta file digambarkan sebagai sebuah pohon, maka pencarian suatu folder atau file dapat dilakukan dengan melakukan traversal graf. Pencarian atau traversal graf dihentikan berdasarkan jenis pencarian. Jika diinginkan pencarian seluruh file dengan nama yang sama, maka traversal dilakukan secara menyeluruh dan berhenti ketika seluruh folder atau file sudah dikunjungi. Sementara itu, jika diinginkan pencarian terhadap salah satu file dengan nama tertentu, maka pencarian akan dihentikan begitu file ditemukan, atau ketika seluruh file dan folder sudah dikunjungi tanpa ada ditemukannya kemunculan file target.

Hasil pencarian menggunakan masing-masing metode disusun sebagai sebuah objek, dengan jenis dan deskripsinya dijelaskan pada tabel berikut

Metode pencarian	Atribut	Jenis Atribut	Deskripsi
BFS	found	boolean	Bernilai true jika setidaknya terdapat sekali kemunculan dari file bernama sama.
	target_name	string	Berisi nama file target.
	queue	Queue of string	Antrian dari simpul/folder yang akan ditelusuri berikutnya.
	pathList	List of string	Berisi alamat-alamat lengkap dari seluruh file yang telah ditemukan.

	tree	Tree	Pohon dinamis dari struktur folder dan file.
	visited	List of string	Berisi alamat-alamat file dan folder yang telah dikunjungi
DFS	found	boolean	Bernilai true jika setidaknya terdapat sekali kemunculan dari file bernama sama.
	target_name	string	Berisi nama file target.
	pathList	List of string	Berisi alamat-alamat lengkap dari seluruh file yang telah ditemukan.
	stack	Stack of string	Tumpukan dari simpul/folder yang akan ditelusuri berikutnya.
	output	List of string	Berisi alamat-alamat file dan folder yang telah dikunjungi
	tree	Tree	Pohon dinamis dari struktur folder dan file.

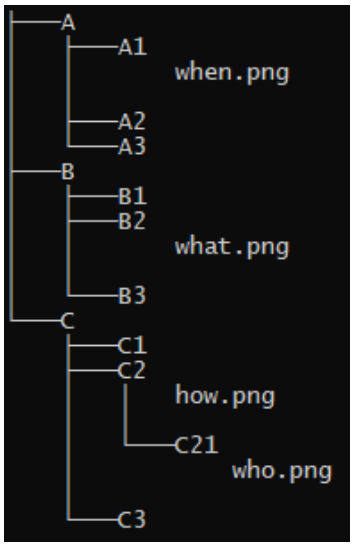
Objek hasil pencarian juga memiliki beberapa metode seperti pada tabel berikut.

Metode Pencarian	**Nama Metode**	**Deskripsi**
BFS	BFS(string path, string target_name, bool findall)	Konstruktor untuk hasil pencarian menggunakan algoritma BFS. Parameter path merupakan alamat dari folder awal, target_name merupakan nama folder yang akan dicari, dan findall menandakan apakah dilakukan pencarian untuk semua kemunculan file atau salah satu kemunculan saja.
	SearchBFS(string path, bool findall)	Fungsi utama yang digunakan untuk melakukan traversal pada struktur folder dan file.
	isFound()	Mengembalikan nilai found dari hasil pencarian.
	BuildTree()	Membangun tree pada directory (Tidak bergantung pada target

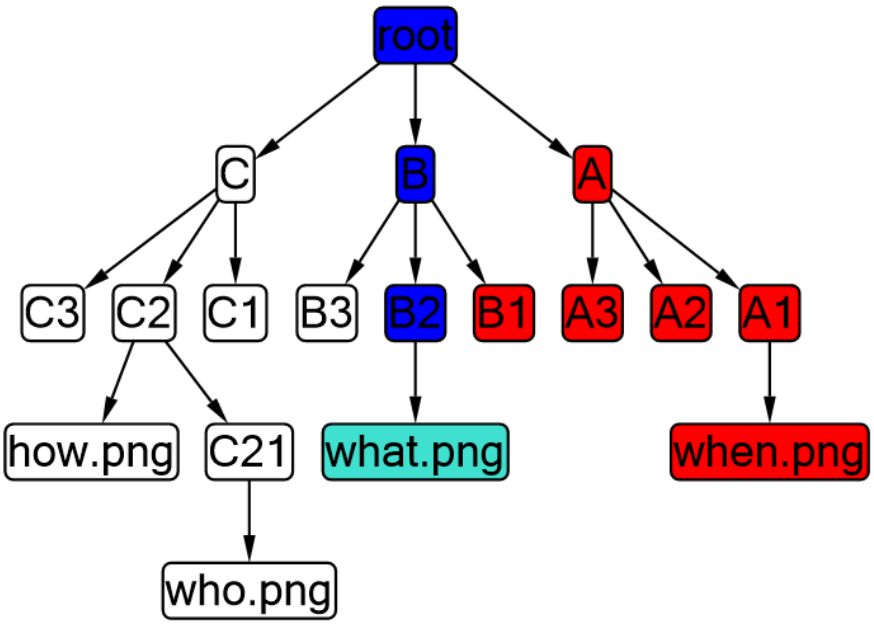
		name) berisi semua folder dan file yang ada di starting path
DFS	DFS(string start_path, string target_name, bool findall)	Konstruktor untuk hasil pencarian menggunakan algoritma DFS. Parameter path merupakan alamat dari folder awal, target_name merupakan nama folder yang akan dicari, dan findall menandakan apakah dilakukan pencarian untuk semua kemunculan file atau salah satu kemunculan saja.
	SearchDFS(string path, bool findall)	Fungsi utama yang digunakan untuk melakukan traversal pada struktur folder dan file.
	isFound()	Mengembalikan nilai found dari hasil pencarian.
	BuildTree()	Membangun tree pada directory (Tidak bergantung pada target name) berisi semua folder dan file yang ada di starting path

Ilustrasi Contoh Kasus

Pengguna ingin mengetahui langkah folder crawling untuk menemukan file what.png. Folder awal pencarian adalah folder *root*, yang memiliki struktur sebagai berikut.



Maka, path pencarian what.png dengan menggunakan metode DFS adalah sebagai berikut.



root → A → A1 → when.png → A1 → A → A2 → A → A3 → A → root → B → B1 → B → B2 → what.png

Pencarian akan terus menerus dilakukan ke dalam salah satu anak simpul, apabila sudah tidak ada anak barulah melakukan backtracking hingga sampai di simpul yang masih memiliki anak yang belum ditelusuri. Hal tersebut terus dilakukan sampai file tujuan ditemukan atau seluruh simpul telah dikunjungi.

Pada visualisasi di atas, path yang telah ditelusuri berwarna merah, path yang menuju file tujuan (what.png) berwarna biru, dan path yang belum ditelusuri berwarna merah.

Bab IV
Implementasi & Pengujian

Implementasi Program

Pseudocode Program Utama

```
main()

Deklarasi Variabel

      path             : string

      f1               : FileNameInput

      folderBrowserDialog : FolderBrowserDialog

      watch            : Stopwatch

Algoritma

      setUpApp()

      folderBrowserDialog ← FolderBrowserDialog()

      f1 ← FileNameInput(path)

      path ← folderBrowserDialog.SelectedPath()

      if (f1.method_choice = 0) then

            if (f1.scope_choice = 1) then

            watch.Start()

            bfs  : BFS

            bfs ← BFS(path, target_name, true)

            watch.Stop()

            f2   : FileFound

            f2 ← FileFound(path, bfs.pathList, bfs.isFound(),
watch.ElapsedMilliseconds)

            runApp(f2)

            bfs.tree.displayTree(bfs.visited, bfs.pathList)

            else

            watch.Start()
```

```
           bfs : BFS

           bfs ← BFS(path, target_name, false)

           watch.Stop()

     ←           f2 : FileFound

           f2 ← FileFound(path, bfs.pathList, bfs.isFound(),
watch.ElapsedMilliseconds)

           runApp(f2)

           bfs.tree.displayTree(bfs.visited, bfs.pathList)

      else

           if (f1.scope_choice = 1) then

           watch.Start()

           dfs  : DFS

           dfs ← DFS(path, target_name, true)

           watch.Stop()

           f2   : FileFound

           f2 ← FileFound(path, dfs.pathList, dfs.isFound(),
watch.ElapsedMilliseconds)

           runApp(f2)

           dfs.tree.displayTree(dfs.visited, dfs.pathList)

           else

           watch.Start()

           dfs  : DFS

           dfs ← DFS(path, target_name, false)

           watch.Stop()

           f2   : FileFound
```

```
            f2 ← FileFound(path, dfs.pathList, dfs.isFound(),
watch.ElapsedMilliseconds)

            runApp(f2)

            dfs.tree.displayTree(dfs.visited, dfs.pathList)
```

Pseudocode Breadth First Search

```
procedure BFS (input start_path: input, input: target_name,
input findall : boolean)


Deklarasi Variabel

     found: bool

     target_name    : string

     queue          : Queue<string>

     start_path     : string

     pathList       : List<string>

     tree           : Tree

     visited        : List<string>



ALGORITMA

     this.start_path = start_path

     this.found = false

     this.target_name = target_name

     BuildTree(start_path)

     queue.Enqueue(start_path)

     if (findall) then

          while (queue.Count > 0) do

               head : string

               head ← queue.Dequeue
```

```
            SearchBFS(head, findall)
    else
        while (not found and queue.Count > 0) do
            head : string
            head ← queue.Dequeue
            SearchBFS(head, findall)


procedure SearchBFS(input path : string, input findall :
bool)


Deklarasi Variabel
    files     : array of string
    folders   : array of string
ALGORITMA
    try
        files ← Directory.GetFiles(path)
        folders ← Directory.GetDirectories(path)
    catch
    finally
        if (files != null) then
            if (findall) then


                i traversal[0..files.Length-1]
```

```
                        if (Path.GetFileName(files[i] =
target_name) then
                              found ← true
                              pathList.Add(files[i])
                        else
                              visited.Add(files[i])
               else
                    i : integer
                    i ← 0
                    while (not found and i < files.Length)
do
                        if (Path.GetFileName(files[i]) =
target_name) then
                              found ← true
                              pathList.Add(files[i])
                        else
                              visited.Add(files[i])
                              i ← i + 1
          if ((findall or (not findall and not found)) and
folders != null) then
               k traversal[0…folders.Length-1]
                    queue.Enqueue(folders[k])
          visited.Add(path)
```

Pseudocode Depth First Search

```
procedure DFS (input start_path: input, input: target_name,
input findall : boolean)


Deklarasi Variabel

     found          : bool

     target_name    : string

     stack          : Stack<string>

     start_path     : string

     pathList       : List<string>

     tree           : Tree

     output         : List<string>



ALGORITMA

     this.start_path = start_path

     this.found = false

     this.target_name = target_name

     BuildTree(start_path)

     stack.Push(start_path)

     SearchDFS(stack.Peek(), findall)


procedure SearchDFS(input path: string, input findall:
boolean)
```

```
Deklarasi Variabel
     files     : array of string
     folders   : array of string
ALGORITMA
     try
          files ← Directory.GetFiles(path)
          folders ← Directory.GetDirectories(path)
     catch
     finally
          if (folders.Length != 0) then
               Array.Sort(folders)
               i : integer
               i ← 0
               while (i < folders.Length and
output.Countains(folders[i])) do
                    i ← i + 1
               if (i != folders.Length) then
                    stack.Push(folders[i])
                    output.Add(folders[i])
                    SearchDFS(stack.Peek(), findall)
               else
                    CheckFileInDirectory(files, findall)
```

```
        else
            CheckFileInDirectory(files, findall)
            if (not found)
                if (stack.Count > 1) then
                    stack.Pop()
                    searchDFS(stack.Peek(), findall)
```

Struktur Data Program

Secara garis besar terdapat dua macam file data program yang digunakan, yaitu Search dan Tree

1. **Search File**

   Search file terdiri atas dua class, yaitu BFS dan DFS.

BFS memiliki atribut berupa:

- found (boolean)
- target_name(string)
- queue(Queue)
- start_path(string)
- pathList(List<string>)
- tree(Tree)
- visited (List<string>)
- inQueue (List<string>)

BFS memiliki method berupa:

- BFS() → Constructor yang menerima start_path, target_name, & boolean findall
- SearchBFS() → Fungsi inti dari algoritma Breadth-First Search
- BuildTree() → Membangun tree dari semua folder/file yang ada di start_path
- isFound() → Mengembalikan true jika sudah found

DFS memiliki atribut berupa:

- found (boolean)
- target_name(string)
- stack(Stack)
- pathList(List<string>)
- tree(Tree)
- output (List<string>)

DFS memiliki method berupa:

- DFS() → Constructor yang menerima start_path, target_name, & boolean findall

- CheckFileInDirectory() → Mengecek semua file yang ada di dalam directory
- SearchDFS() → Fungsi inti dari algoritma Depth-First Search
- BuildTree() → Membangun tree dari semua folder/file yang ada di start_path
- isFound() → Mengembalikan true jika sudah found

2. Tree File

File Tree berisi class Node dan class Tree. File ini berisi implementasi semua fungsi yang digunakan untuk membangun sebuah tree dan menampilkannya pada GUI. Terdapat beberapa method-method penting yang digunakan pada main program seperti:

- addChild() → Membuat child baru pada tree
- updateParentColor() → Mewarnai parent node
- displayTree() → Menampilkan tree dengan warna yang berbeda-beda tergantung status nodenya (Visited, InQueue, Solution Path, atau Solution node)

Tata Cara Penggunaan Program

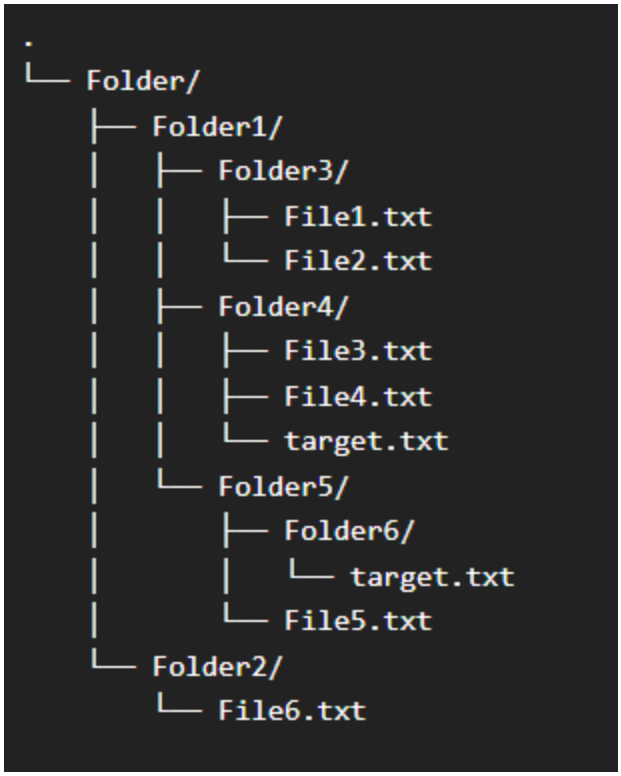
Untuk dapat menjalankan program, unduh terlebih dahulu program yang terdapat pada repository GitHub berikut ini.

https://github.com/NayotamaPradipta/PencerahanExplorer.git

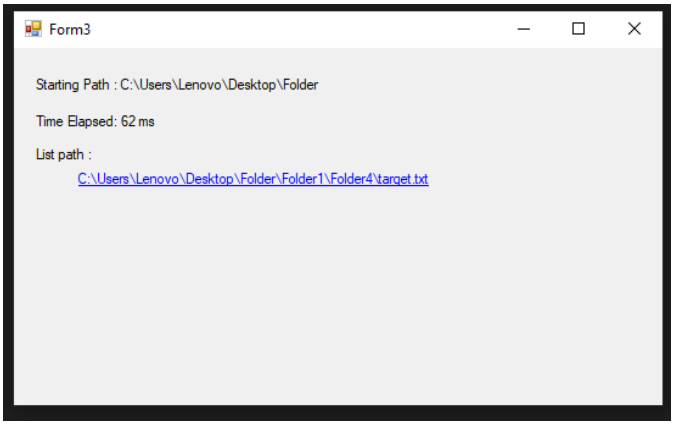
Kemudian, buka file solution bernama PencerahanExplorer.sln pada Microsoft Visual Studio, lalu klik ikon start untuk menjalankan program pencarian. Pastikan bahwa pada perangkat yang digunakan telah terinstal modul MSAGL. Panduan untuk melakukan instalasi modul dan pustaka yang perlu diunduh serta cara menjalankan program juga terdapat pada bagian README.md dalam repository.

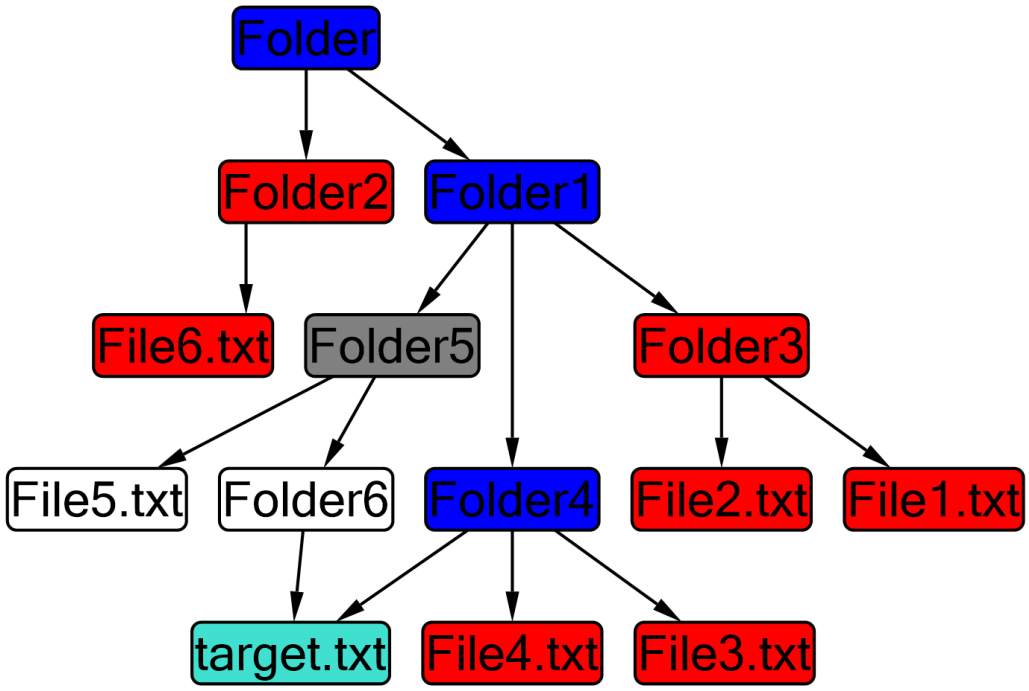
Hasil Pengujian

Input Directory:



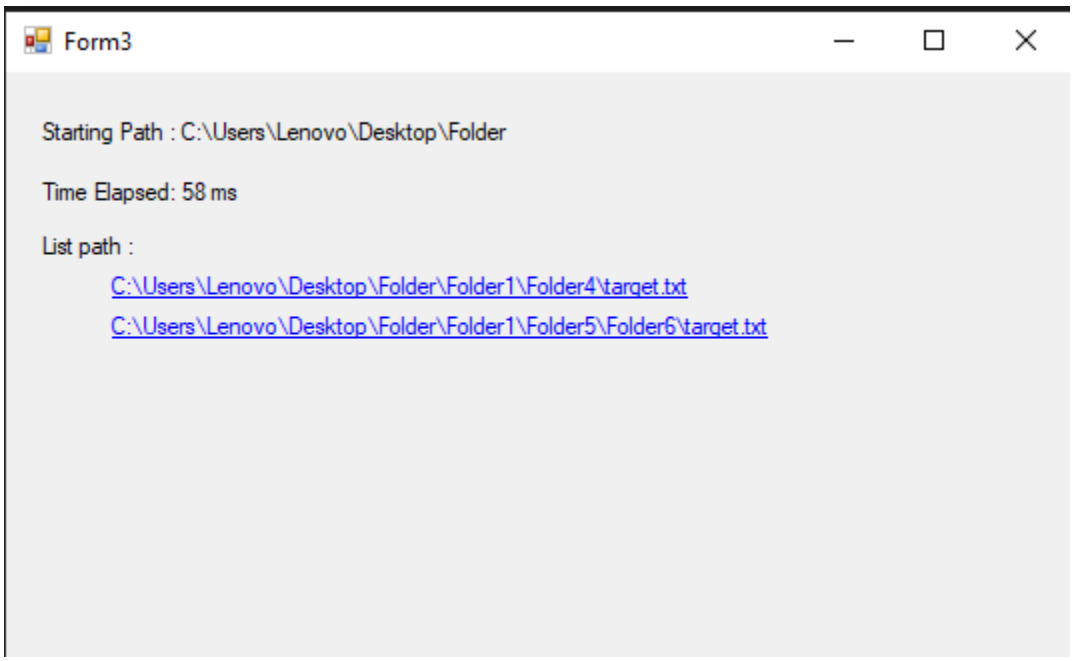
1. **BFS Find One (Target: target.txt)**
   OUTPUT:

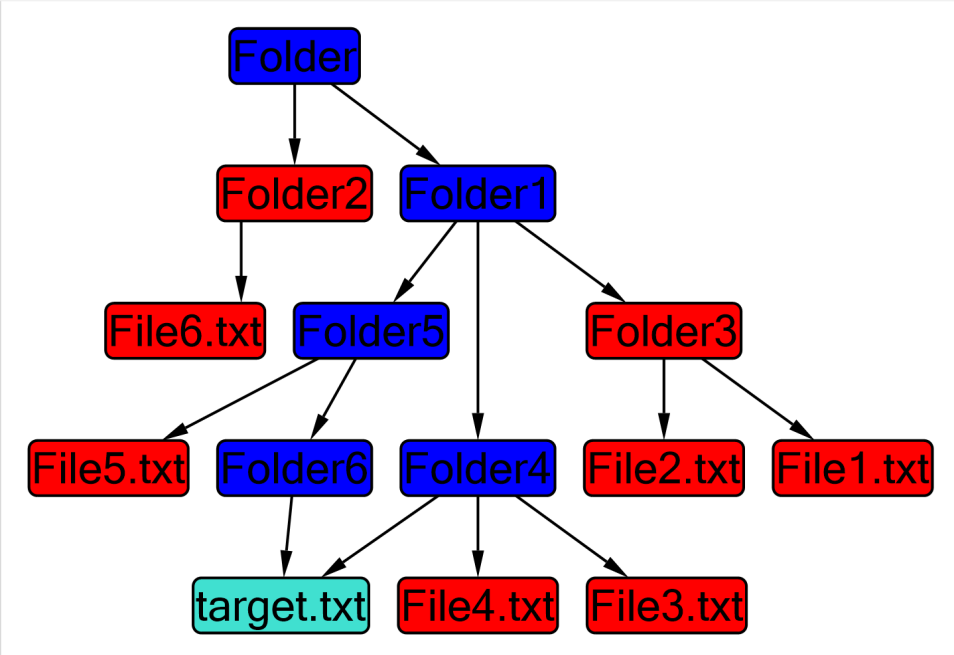




2. **BFS Find All (Target: target.txt)**

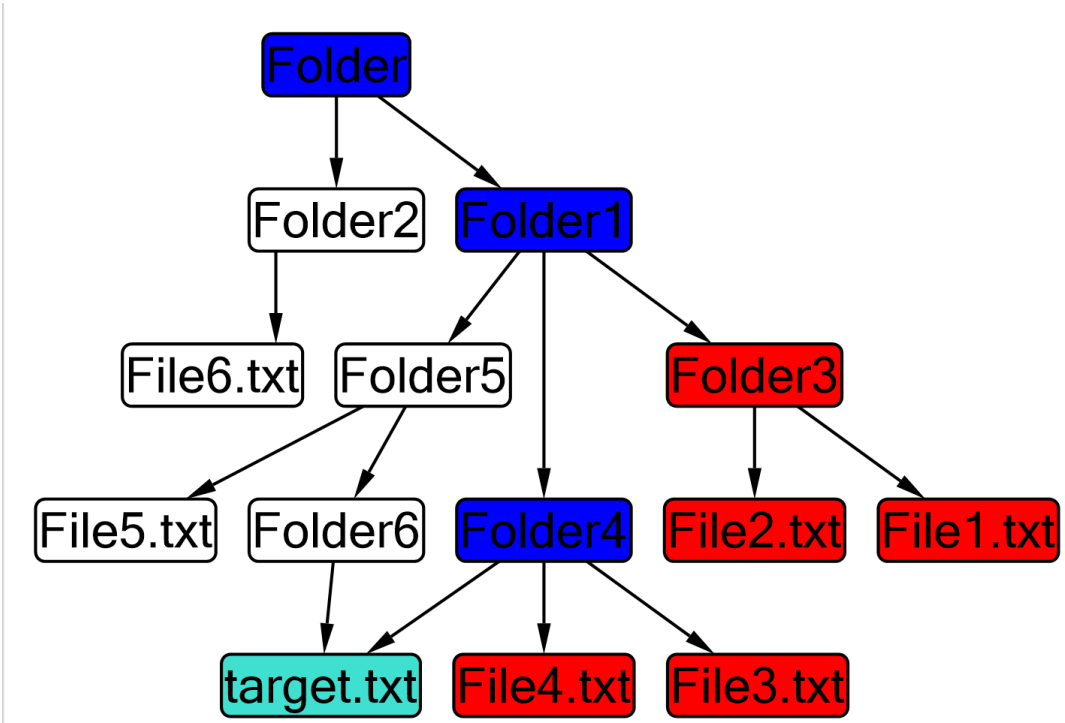
   Output:





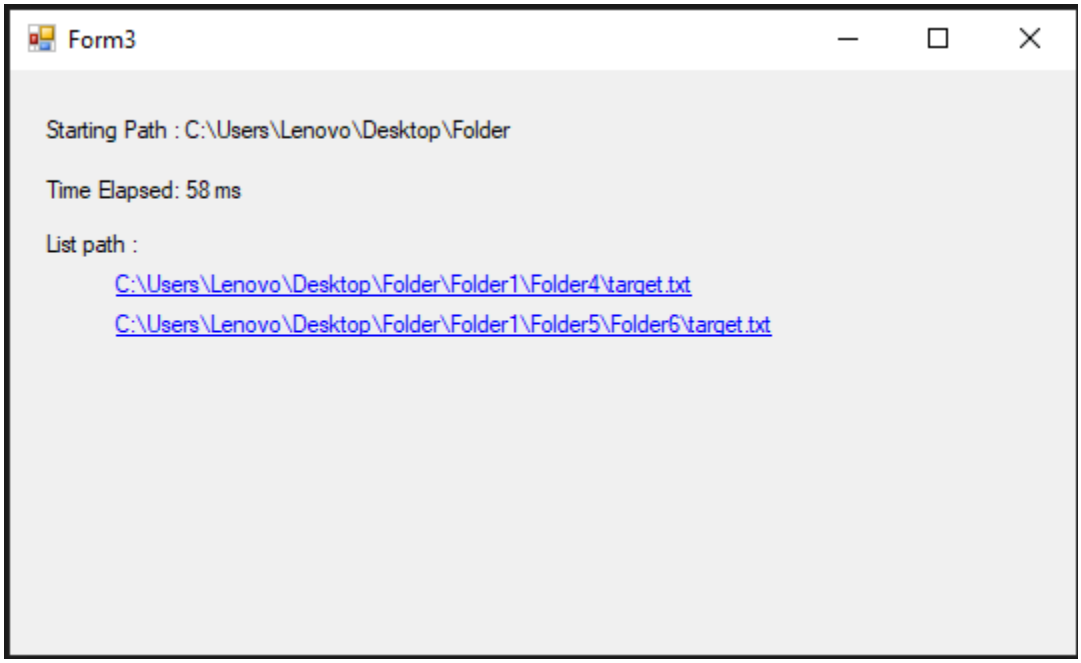
3. **DFS Find One (Target: target.txt)**

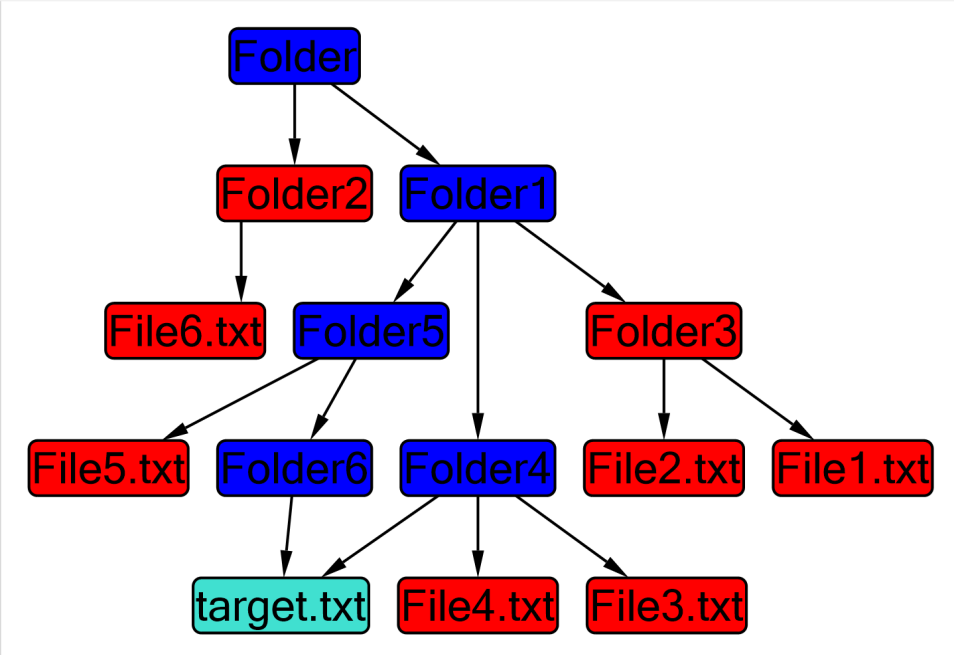
Output:



4. **DFS Find All (Target: target.txt)**

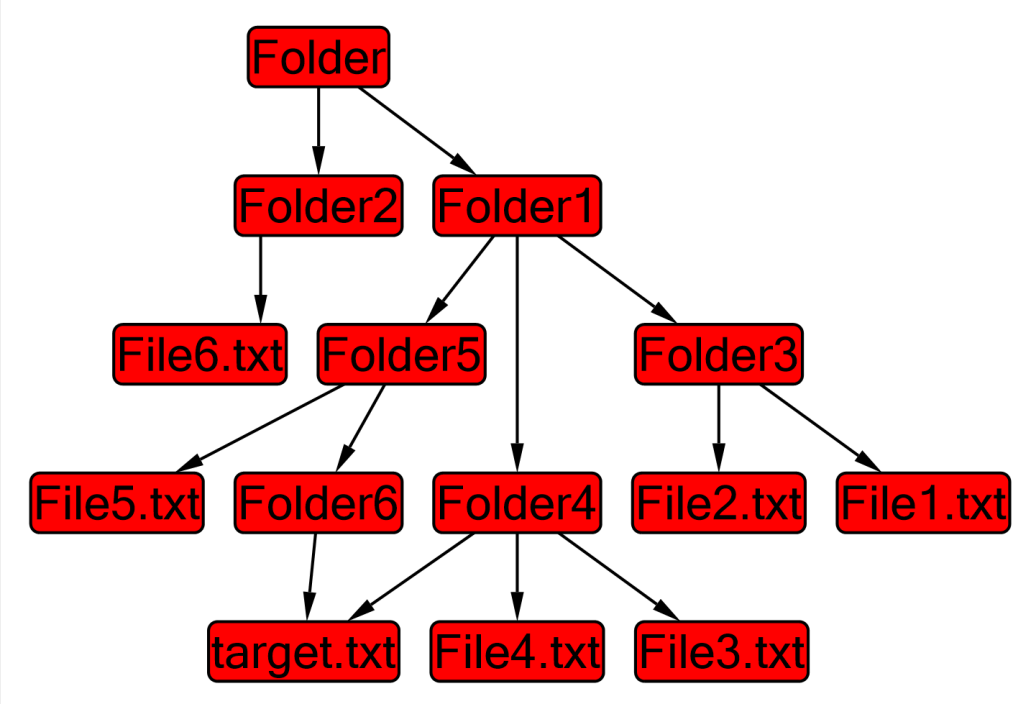
Output:





5. **DFS/BFS Find One / Fine All (Target: Yang tidak ada di directory)**

Output:

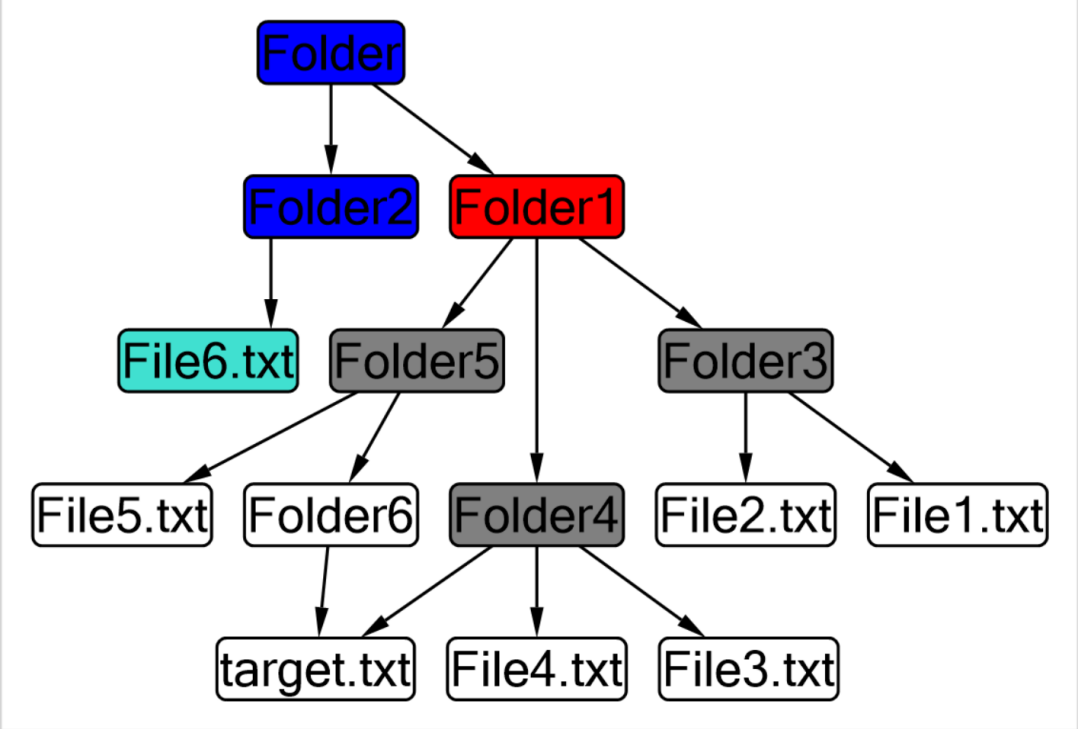


Keterangan:

1. Warna merah berarti folder/file yang telah di visit atau dicek akan tetapi tidak menuju ke solusi
2. Warna biru menunjukkan path menuju solusi
3. Warna turquoise menunjukkan file target/solusi
4. Warna abu-abu menunjukkan folder yang telah dimasukkan ke dalam queue akan tetapi tidak dicek karena file sudah ditemukan pada folder sebelumnya

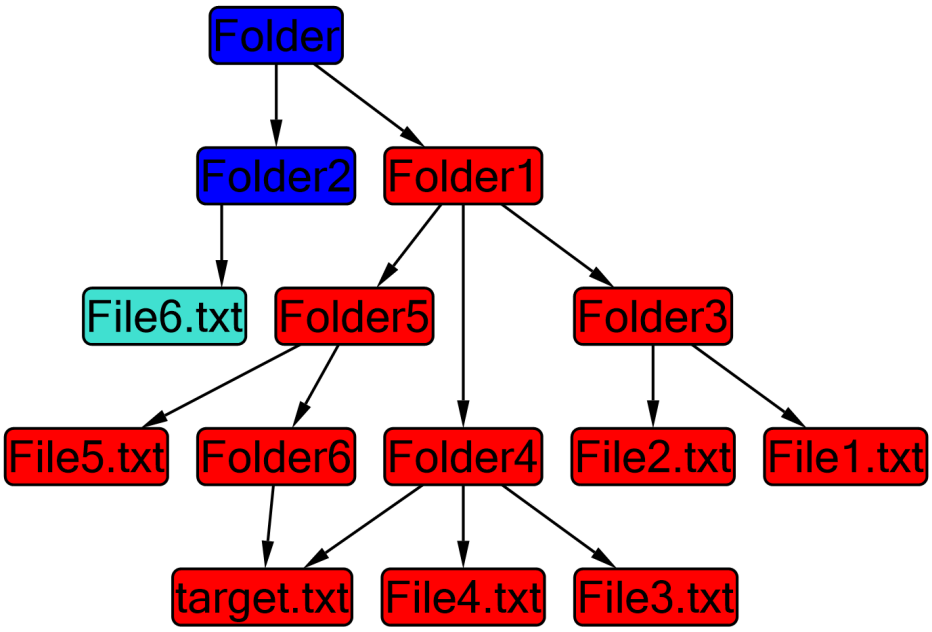
Analisis Desain Solusi Algoritma

Breadth-First Search dan Depth-First Search merupakan algoritma yang baik digunakan dalam Folder Crawling, terutama jika dibandingkan dengan Brute Force Algorithm. Keduanya memiliki kekuatan dan kelemahan masing-masing. Breadth-First Search Algorithm akan sangat efektif jika file yang ingin dicari berada di tingkatan yang cukup rendah. Sebagai contoh, untuk folder Input di atas, BFS akan menjadi efektif untuk mencari File6.txt. Hal ini dapat dibuktikan pada tree dibawah:



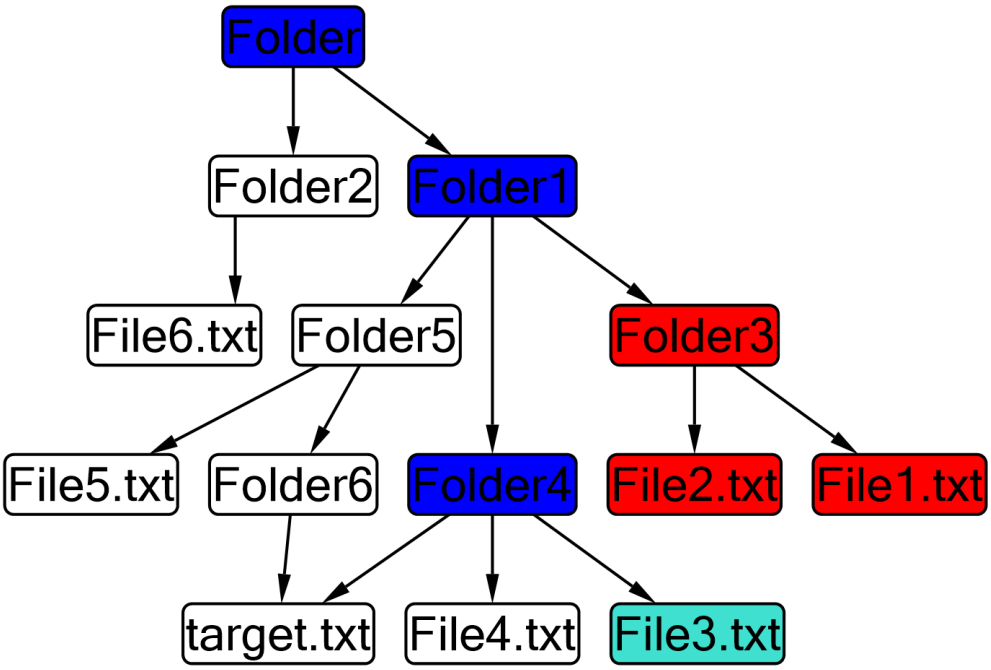
Tampak bahwa pada BFS, program hanya perlu untuk mengecek folder1, kemudian folder2, lalu langsung didapat solusinya.

Di sisi lain, pencarian "File6.txt" menggunakan DFS akan membutuhkan waktu yang sangat lama, bahkan paling lama. Buktinya sebagai berikut:

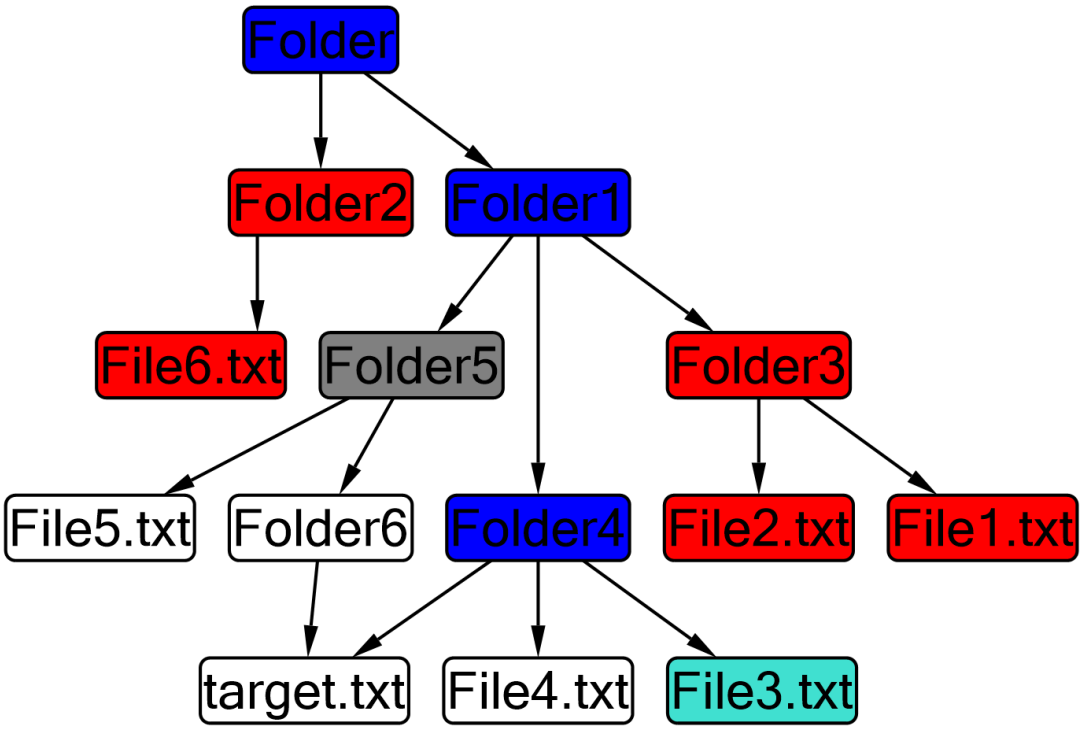


Tampak bahwa untuk mencari "File6.txt" menggunakan DFS, program perlu mengecek keseluruhan direktori sehingga membutuhkan waktu yang jauh lebih lama.

DFS akan menjadi bagus apabila file berada di tingkatan yang “dalam” karena bisa jadi langsung ketemu pada pencarian pertama. Hal ini dapat dibuktikan pada tree diagram berikut:



Untuk mencari File3.txt, terdapat tiga node yang berwarna merah. Hal ini lebih sedikit jika dibandingkan dengan BFS yang memiliki lima node berwarna merah, sebagai berikut:



Kedua skenario sangat menjelaskan kelebihan dan kekurangan untuk BFS dan DFS. Sebagai kesimpulan, BFS akan jauh lebih baik dan efisien jika folder tidak bercabang terlalu dalam, sedangkan DFS akan lebih baik jika directory memiliki tingkatan yang banyak.

Bab V
Kesimpulan & Saran

Kesimpulan

Pada tugas besar IF2211 Strategi Algoritma Semester 2 berjudul “Implementasi Folder Crawling”, kelompok kami, Pencerahan Explorer, berhasil memanfaatkan algoritma Breadth-First Search & Depth-First Search menggunakan bahasa C#. Baik BFS maupun DFS memiliki dua macam algoritma, yaitu mencari satu file dan mencari semua file. Masing-masing algoritma memiliki kelebihan dan kelemahan yang telah dijelaskan pada bab sebelumnya. Setelah mengerjakan tugas besar ini, kami telah memperdalam pengetahuan mengenai C#, BFS, beserta DFS karena langsung diimplementasikan pada permasalahan yang nyata.

Saran

Setelah menjalankan tugas besar ini, saran yang dapat kami berikan untuk tugas besar IF2211 Strategi Algoritma semester 2 adalah:

1. Memperjelas spesifikasi dan batasan-batasan untuk menghindari adanya multitafsir
2. Penulisan pseudocode kurang efektif karena sebenarnya programnya sangat panjang sehingga tidak semua fungsi pada program kami tuliskan dalam bentuk pseudocode (Asumsi sudah ada fungsinya tinggal pakai)
3. Memberikan tutorial atau introduction mengenai C# dan Visual Code agar tidak mulai dari nol, karena pada kurikulum ITB sendiri kami tidak diajarkan C# sama sekali, apalagi Windows Development

Daftar Pustaka

1. Munir, Rinaldi. 2022. Slide Kuliah Strategi Algoritma - Semester II Tahun 2021/2022, diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/stima21-22.htm#SlideKuliah

Tautan Terkait

Link Repository Github

https://github.com/NayotamaPradipta/PencerahanExplorer.git

Link Video Demonstrasi Program

https://youtu.be/d5RwA7zwlf4