TUGAS BESAR 2 IF2211
STRATEGI ALGORITMA

oleh

Rozan Fadhil Al Hafidz	13520039
Andreas Indra Kurniawan	13520091
Zayd Muhammad Kawabiki Zuhri	13520144

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

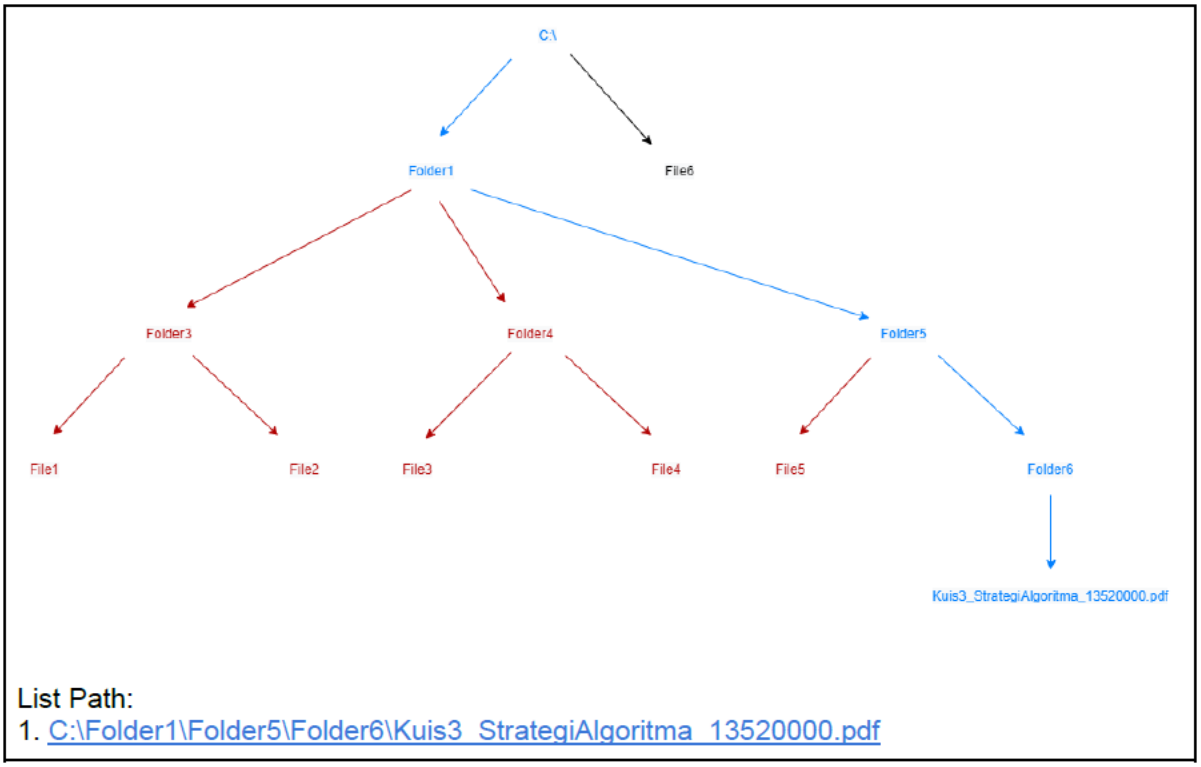
2022

Daftar Isi

Bab 1
Deskripsi Tugas

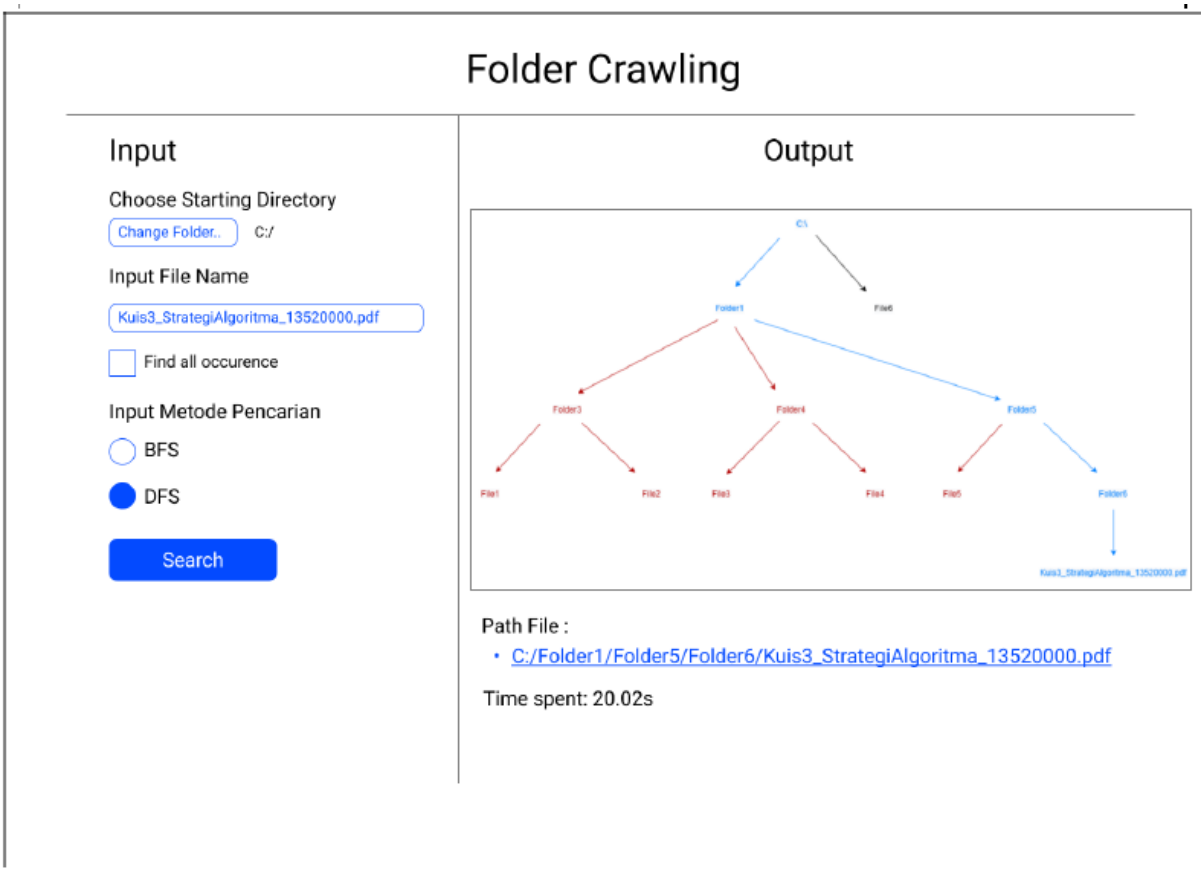
Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari *file explorer* pada sistem operasi, yang pada tugas ini disebut dengan *Folder Crawling*. Dengan memanfaatkan algoritma *Breadth First Search* (BFS) dan *Depth First Search* (DFS),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



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.

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



BAB 2
LANDASAN TEORI

1. Dasar Teori

a. Graph Traversal

Graf merupakan suatu representasi data berupa objek-objek diskrit yang disusun secara terstruktur dengan hubungan-hubungan atau keterkaitan tertentu. Suatu graf umumnya terdiri dari dua elemen, yaitu simpul (vertex/node) dan sisi (edge). Simpul menggambarkan objek-objek diskrit dalam persoalan, sedangkan sisi menggambarkan hubungan antar dua pasangan objek. Dalam persoalan tugas besar ini, graf digunakan untuk merepresentasikan struktur *file* suatu *directory* yang menghasilkan suatu *tree*. Setiap simpul merupakan folder, dengan daun-daun *tree* merupakan *file*, dan sisi merepresentasikan dua folder yang terhubung (atau file yang berada di dalam suatu folder).

Traversal graf merupakan suatu algoritma yang mencari solusi dengan cara mengunjungi simpul-simpul secara sistematis. Pengunjungan simpul ini bertujuan untuk mengecek, membandingkan, mencari, atau mengubah objek-objek dalam representasi graf tersebut, dengan harapan mendapatkan solusi persoalan. Terdapat beberapa klasifikasi traversal graf. Yang pertama adalah berdasarkan informasi yang tersedia dalam konteks graf yang ditinjau. *Informed search* dilakukan jika terdapat informasi tambahan, sehingga pencarian dilakukan berdasarkan aturan heuristik. Contohnya adalah *A* Algorithm* dan *Best First Search*. Sedangkan pada *Uninformed/blind search,* tidak ada informasi yang tersedia, sehingga pencarian harus mengikuti aturan seperti pada DFS, BFS, DLS, dll. Contoh tersebut juga merupakan klasifikasi traversal graf selanjutnya, yaitu berdasarkan urutan pengunjungan simpul-simpul pada graf seperti perbedaan antara Breadth First Search (BFS) dan Depth First Search (DFS).

b. BFS

Breadth First Search atau disingkat menjadi BFS, merupakan salah satu algoritma traversal graf yang bersifat uninformed. Sesuai dengan namanya, BFS mengikuti aturan pengunjungan simpul yang bersifat melebar. Dengan kata lain, algoritma ini memprioritaskan penelusuran setiap lapisan graf secara lengkap terlebih dahulu. Pada tree akan terlihat jelas bahwa BFS akan mengunjungi semua simpul pada suatu kedalaman terlebih dahulu sebelum menulusuri kedalaman yang lebih dalam.

Secara singkat, algoritma BFS berjalan sebagai berikut. Jika traversal dimulai pada suatu simpul S, maka kunjungi simpul S tersebut terlebih dahulu. Lalu, kunjungi setiap simpul yang terhubung dengan simpul S, alias semua tetangga simpul S terlebih dahulu. Setelah itu kunjungi setiap simpul yang berhubungan dengan semua simpul yang dikunjungi tadi, dengan aturan FIFO atau First In First Out, di mana simpul yang dikunjungi lebih awal juga akan dikunjungi tetangganya lebih awal. Lakukan ini sampai ditemukan solusi persoalan atau seluruh simpul dalam graf sudah dikunjungi.

c. DFS

Depth First Search yang disingkat menjadi DFS, merupakan alternatif lain dalam algoritma traversal graf. Sesuai namanya, DFS berbeda dengan BFS karena mengikuti aturan pengunjungan simpul yang bersifat mendalam. DFS menelusuri graf dengan memprioritaskan pengunjungan simpul yang berada di lapisan lebih dalam, lalu melakukan backtrack jika pada ujungnya tidak menemukan solusi.

Secara singkat, algoritma DFS berjalan sebagai berikut. Jika traversal dimulai pada suatu simpul S, maka kunjungi suatu simpul T yang bertetangga dengan simpul S. Lalu, kunjungi suatu simpul yang bertetangga dengan simpul T, dan seterusnya diulangi. Ketika pengujungan mencapai suatu

simpul Usedemikian hingga tidak ada lagi simpul tetangga dari U yang belum dikunjungi, dilakukan *backtracking* ke simpul sebelumnya yang mempunyai tetangga yang belum dikunjungi, lalu kunjungi simpul tersebut. Lakukan ini sampai ditemukan solusi persoalan atau seluruh simpul dalam graf sudah dikunjungi

2. C# Desktop Application Development

Bahasa pemrograman C# atau juga dibaca 'C sharp' merupakan bahasa pemrograman yang dikembangkan oleh Microsoft mulai pada tahun 2000 dan digunakan untuk mengembangkan web apps, desktop apps, game development, dan banyak lainnya. Bahasa ini sangat erat terkait dengan konsep object-oriented programming, dengan fitur seperti inheritance, class, polymorphism, dll. Microsoft mengembangkan C# dengan environment yang lengkap dengan IDE yaitu Visual Studio dengan Framework .NET. Untuk memudahkan pembuatan GUI pada tugas besar ini, kami menggunakan template Windows Forms Application pada Visual Studio, dengan fitur yang lengkap untuk menambahkan elemen-elemen GUI dan menghubungkannya dengan algoritma BFS dan DFS.

BAB 3
ANALISIS PEMECAHAN MASALAH

1. Langkah-Langkah Pemecahan Masalah

Berdasarkan permasalahan yang telah dijelaskan pada Bab 1 Deskripsi Tugas, kami memecahkan masalah tersebut dengan langkah-langkah sebagai berikut.

1) Memahami permasalahan yang terdapat pada Bab 1 Deskripsi Tugas
2) Memetakan permasalahan menjadi elemen-elemen BFS dan DFS
3) Mempelajari bahasa pemrograman C# dan IDE Visual Studio
4) Membuat program untuk membaca file dan folder secara BFS dan DFS
5) Mengimplementasikan program untuk mencari file secara BFS dan DFS
6) Mendesain GUI (graphical user interface) pada program tersebut
7) Menggabungkan program yang telah dibuat dengan GUI tersebut sehingga tombol-tombol tersebut bisa digunakan dengan semestinya
8) Membuat visualisasi graf pada GUI mengunakan *library* MSAGL
9) Mengimplementasikan tampilan progres pembentukan pohon
10) Melakukan pengujian GUI yang telah dibuat

Dalam melakukan pencarian file dengan algortma DFS, kami membuat program yang menjalankan langkah-langkah berikut

1) Buka folder awal
2) Cek setiap nama file yang ada di folder tersebut
3) Jika tidak ditemukan, pilih satu folder yang berada di urutan teratas berdasarkan abjad
4) Cek setiap nama file yang ada di subfolder tersebut
5) Jika masih tidak ditemukan, ulangi langkah dari no 3 pada folder saat ini
6) Jika tidak ada subfolder lagi, lakukan *backtrack* ke folder sebelumnya, kemudian ulangi langkah no 3
7) Pencarian selesai jika semua folder beserta cabang-cabangnya sudah dicek

Sedangkan dalam melakukan pencarian file dengan algortma BFS, kami membuat program yang menjalankan langkah-langkah berikut

1) Buka folder awal
2) Cek setiap nama file yang ada di folder tersebut
3) Jika tidak ditemukan, masukkan seluruh subfolder yang ada di folder tersebut ke dalam antrian, terurut berdasarkan abjad
4) Ambil folder pada antrian sebagai folder yang akan diaksekusi pertama, kemudian ulangi langkah dari no 2 pada folder tersebut
5) Lakukan sampai tidak ada folder yang tersisa di antrian
6) Pencarian selesai jika tidak ada lagi folder di dalam antrian

2. Proses Mapping Persoalan

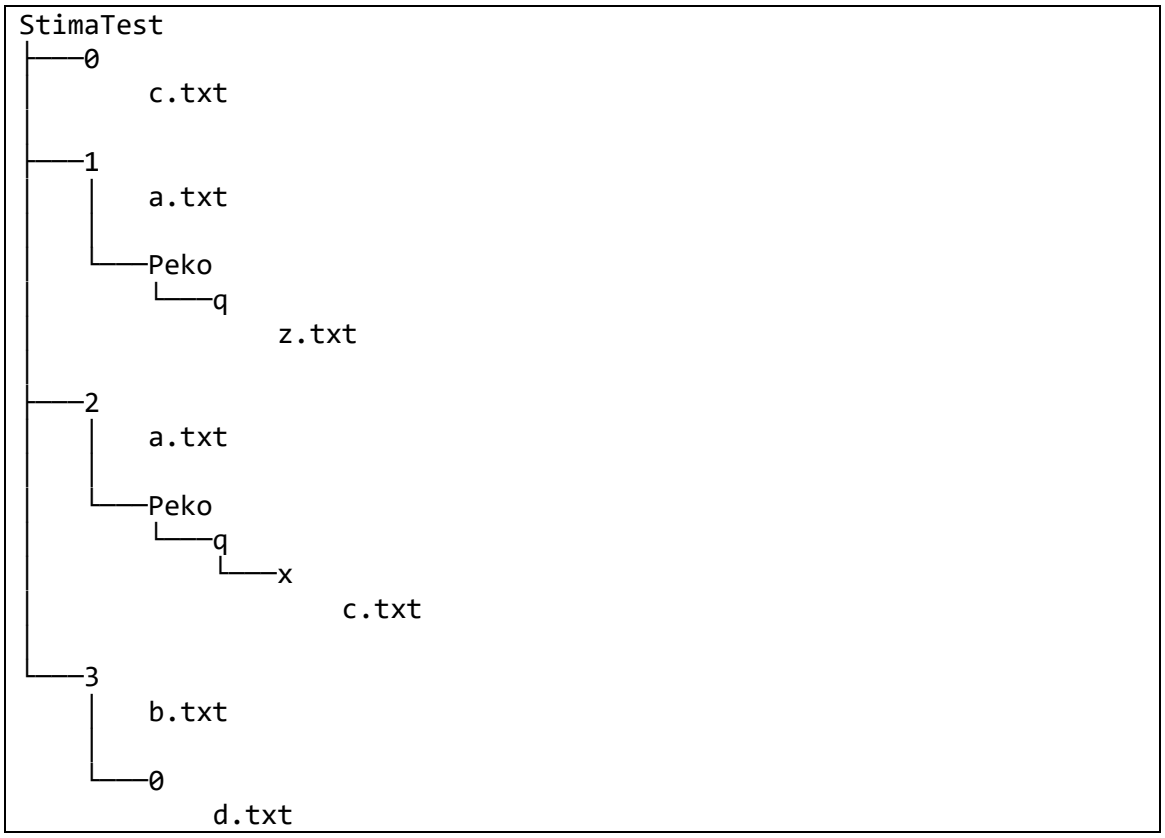
Misalkan dilakukan pencarian file bernama X di dalam sebuah folder bernama F, maka persoalan tersebut akan di-*mapping* sebagai berikut

- Problem state : Mencari folder bernama X pada folder F beserta anak-anaknya
- Akar : Folder F
- Simpul dalam : Folder yang terdapat pada folder F

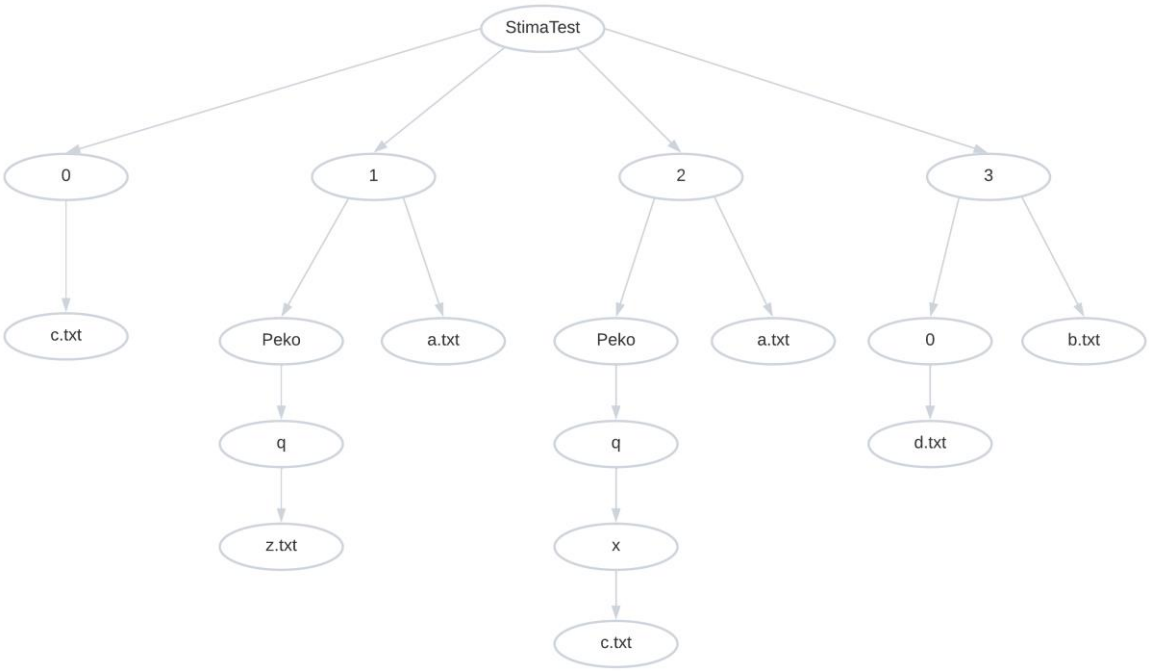
- Daun : File atau folder kosongx yang terdapat pada folder F
- Cabang : Sub-folder dari folder *parent*-nya
- State space : Semua objek yang ada pada folder X, baik file maupun folder
- Solution space : himpunan semua file bernama X

3. Ilustrasi Kasus

Misalkan program ditugaskan untuk mencari seluruh file bernama c.txt di dalam folder StimaTest. Berikut ini adalah isi dari folder StimaTest.



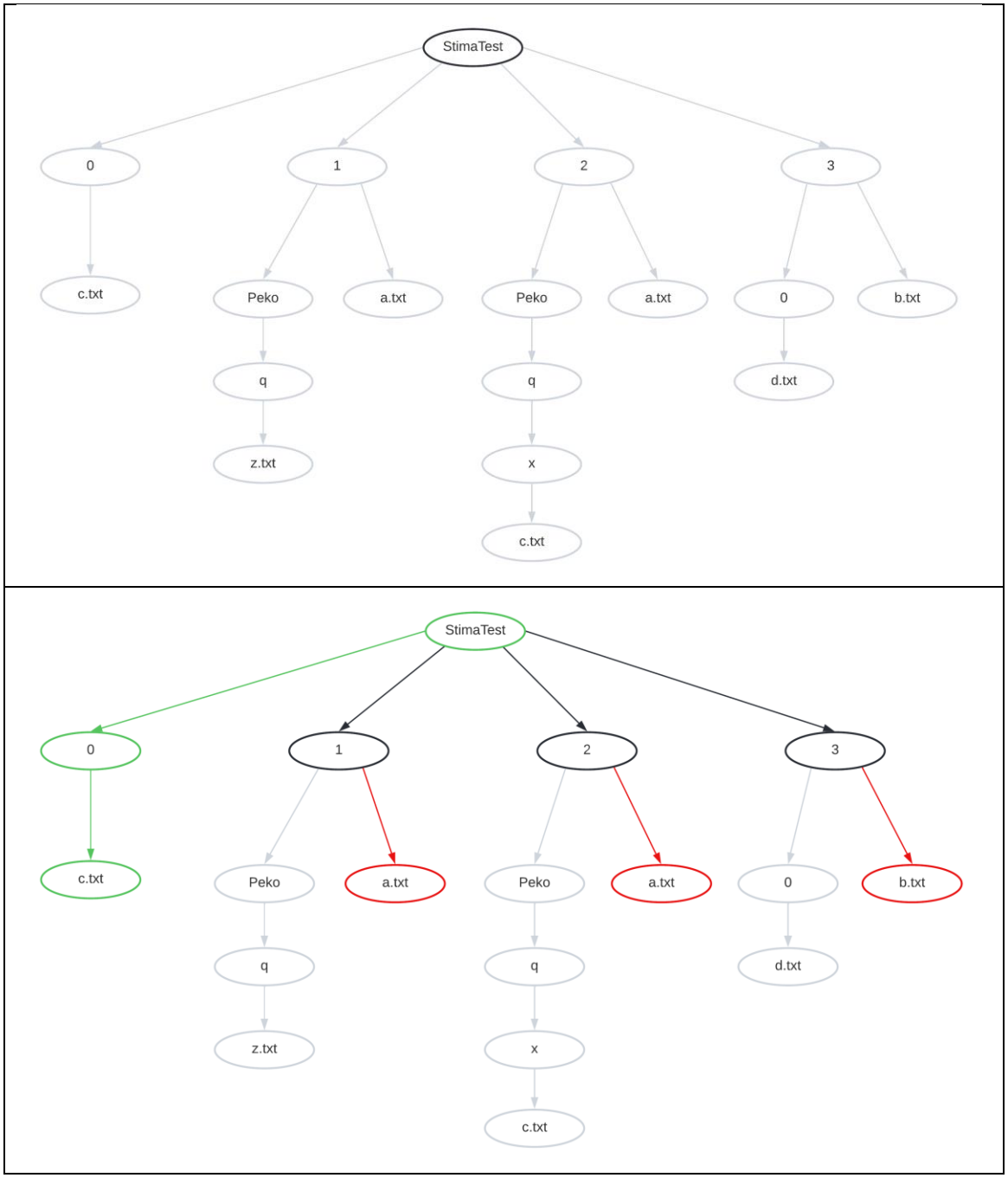
Jika divisualisasikan dengan graf, maka akan terbentuk pohon sebagai berikut.



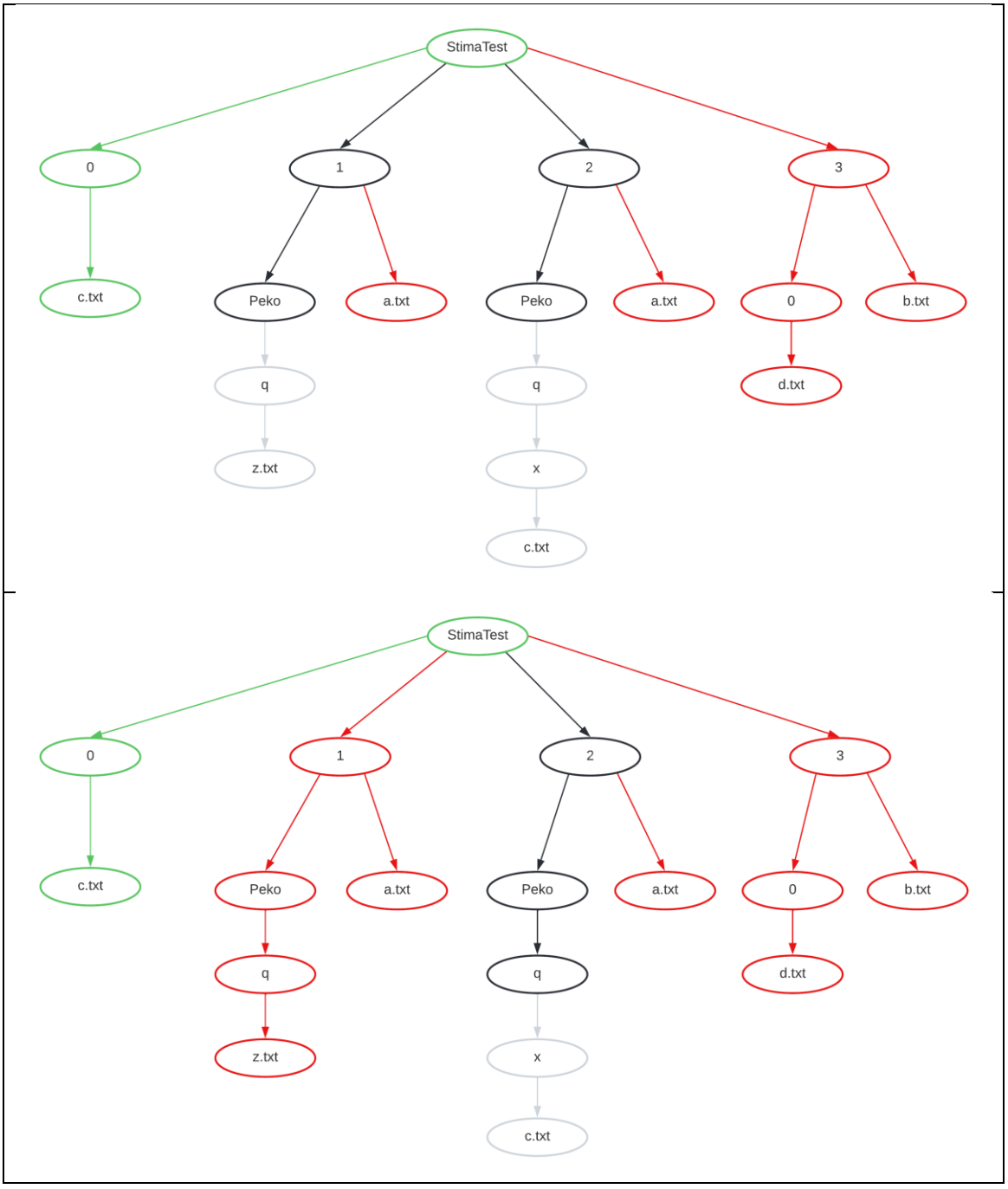
3.1. Pencarian dengan BFS

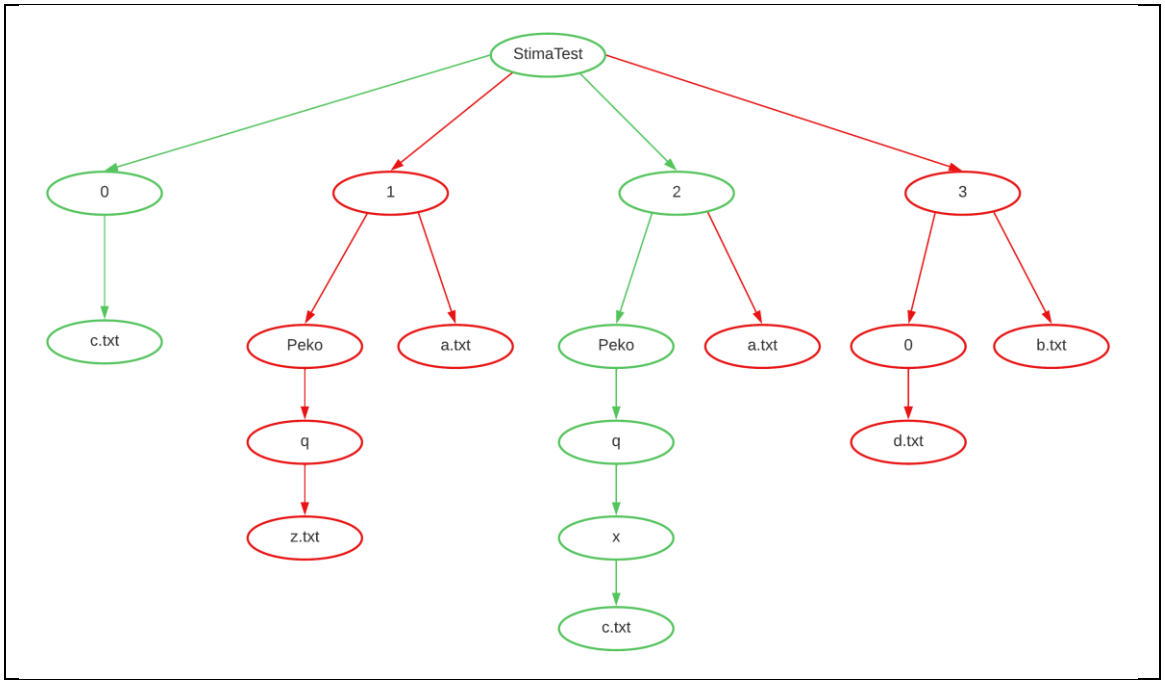
Untuk mencari file dengan menggunakan algoritma BFS, program akan melakukan pengecekan seluruh file yang terdapat di dalam folder StimaTest terlebih dahulu. Setelah itu, program melakukkan pencarian secara melebar, yaitu mengecek seluruh subfolder-subfolder pada folder StimaTest sebelum mengecek anak-anak dari subfolder tersebut. Jika sudah, barulah program akan mengecek anak-anak dari subfolder tersebut. Proses ini dilakukan sampai tidak ada subfolder yang bisa dicek lagi.

Berikut ini adalah ilustrasi pengecekan dengan menggunakan algoritma BFS. Simpul berwarna hitam menandakan bahwa simpul tersebut telah dicek. Simpul berwarna merah menandakan bahwa tidak ada file yang dicari pada simpul tersebut beserta anak-anaknya. Simpul berwarna hijau menandakan simpul yang mengandung atau menuju ke file yang dicari.


StimaTest
0
1
2
3
c.txt
Peko
a.txt
Peko
a.txt
0
b.txt
q
d.txt
z.txt
x
c.txt

StimaTest
0
1
2
3
c.txt
Peko
a.txt
Peko
a.txt
0
b.txt
q
d.txt
z.txt
x
c.txt


StimaTest
0
1
2
3
c.txt
Peko
a.txt
Peko
a.txt
0
b.txt
q
q
d.txt
z.txt
x
c.txt
StimaTest
0
1
2
3
c.txt
Peko
a.txt
Peko
a.txt
0
b.txt
q
q
d.txt
z.txt
x
c.txt

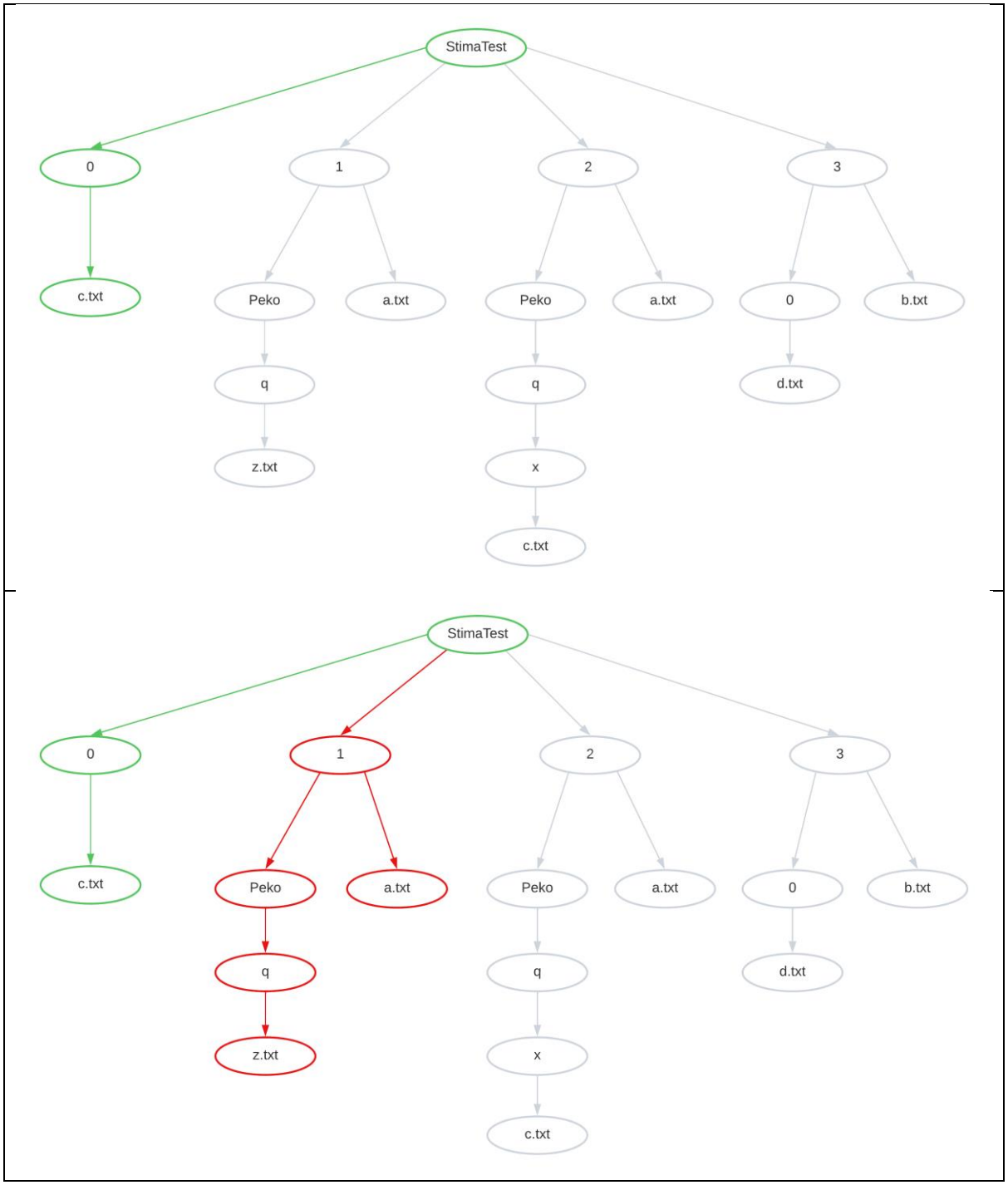


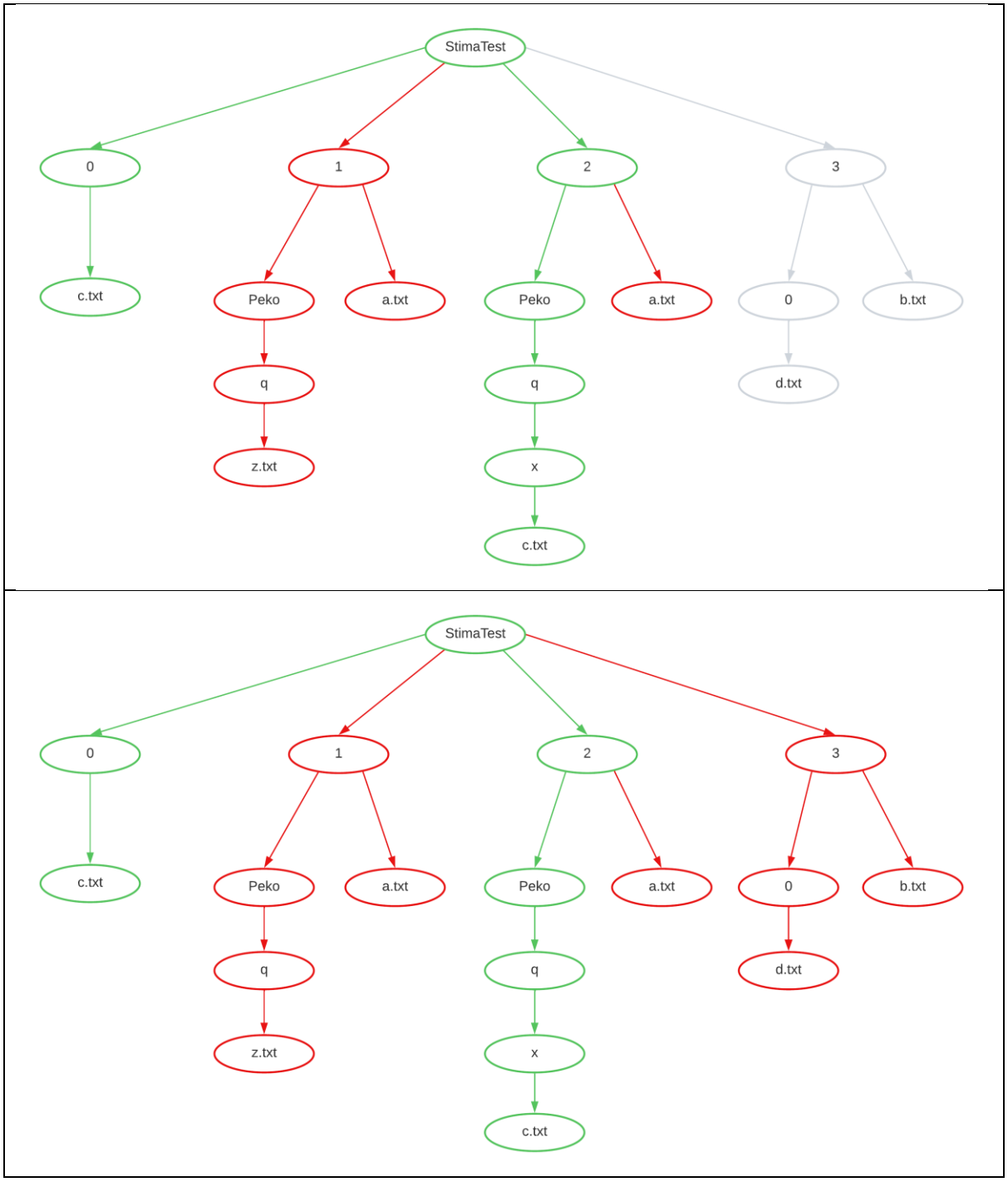
Setelah ditemukan pencarian, ditemukan dua file bernama c.txt, yaitu di StimaTest\0\c.txt dan StimaTest\2\q\x\c.txt. Urutan pengecekannya adalah StimaTest -> 0 -> 1 -> 2 -> 3 -> c.txt -> 0 -> 1 -> a.txt -> Peko -> q -> z.txt -> q -> Peko -> 1 -> StimaTest -> 2 -> a.txt -> Peko -> q -> x -> c.txt -> x -> q -> Peko -> 2 -> StimaTest -> 3 -> b.txt -> 0 -> d.txt

3.2. Pencarian dengan DFS

Untuk mencari file dengan menggunakan algoritma DFS, program juga akan melakukan pengecekan seluruh file yang terdapat di dalam folder StimaTest terlebih dahulu. Setelah itu, program melakukkan pencarian secara mendalam, yaitu memilih sebuah folder yang merupakan subfolder dari folder StimaTest (terurut berdasarkan abjad), kemudian melakukan pengecekan terhadap subfolder tersebut. Jika tidak ditemukan, pilih sebuah anak dari subfolder tersebut (terurut berdasarkan abjad) kemudian lakukan pencarian pada anak tersebut. Jika sudah mencapai simpul daun (tidak ditemukan subfolder lagi), lakukan runut-balik (*backtrack)* ke simpul sebelumnya yang masih memiliki subfolder. Pencarian berakhir ketika seluruh folder yang ada telah dicek.

Berikut ini adalah ilustrasi pengecekan dengan menggunakan algoritma DFS. Sama seperti sebelumnya, simpul berwarna hitam menandakan bahwa simpul tersebut telah dicek. Simpul berwarna merah menandakan bahwa tidak ada file yang dicari pada simpul tersebut beserta anak-anaknya. Simpul berwarna hijau menandakan simpul yang mengandung atau menuju ke file yang dicari.


StimaTest
0
1
2
3
c.txt
Peko
a.txt
Peko
a.txt
0
b.txt
q
q
d.txt
z.txt
x
c.txt
StimaTest
0
1
2
3
c.txt
Peko
a.txt
Peko
a.txt
0
b.txt
q
q
d.txt
z.txt
x
c.txt



Setelah ditemukan pencarian, ditemukan dua file bernama c.txt, yaitu di StimaTest\0\c.txt dan StimaTest\2\q\x\c.txt. Urutan pengecekannya adalah StimaTest -> 0 c.txt -> 0 -> StimaTest -> 1 -> Peko -> q -> z.txt -> q -> Peko -> 1 -> a.txt -> 1 -> StimaTest -> 2 -> Peko -> q -> x -> c.txt -> x -> q -> Peko -> 2 -> a.txt -> 2 -> StimaTest -> 3 -> 0 -> d.txt -> 0 -> 3 -> b.txt -> 3 -> StimaTest

BAB 4
IMPLEMENTASI DAN PENGUJIAN

1. Implementasi Program (*pseudocode program utama)*

a. Variabel Global

```
Global.isRunning : boolean
{ Digunakan dalam pencarian DFS. Bernilai true di awal.
  Jika sudah menemukan solusi, bernilai false untuk memberhentikan pencarian rekursif }


Global.pathQueue : list of string
{ Digunakan dalam pencarian BFS untuk menyimpan antrian path yang akan dicari.
  Antrian path merupakan subfolder-subfolder dari folder utama yang ditemukan selama
pencarian }

Global.result : list of string
{ Digunakan dalam pencarian BFS dan DFS untuk menyimpan path file yang ditemukan }
```

b. Fungsi Pembantu

```
procedure colorEdge(path : string, color : Color)
{ Digunakan untuk mengubah warna edge berhubungan dengan path sesuai color
  Initial State : edge yang berhubungan dengan path terdefinisi
  Final State : edge yang berhubungan dengan path berwarna sesuai color }

function copyList(l : list of string) -> list of string
{ Digunakan untuk mengkopi list
  Masukan : l = list of string
  Keluaran : hasil copy dari list l }
```

c. Algoritma DFS

```
function searchDFS(fileToSearch, path : string, searchAll : boolean, graph : Graph,
timeDelay : int) -> list of string
{ Melakukan inisiasi pencarian DFS. Global.result adalah larik string yang merupakan
variabel Global
  (bisa diakses di mana saja).

  Masukan : fileToSearch = nama file yang akan dicari
            path = lokasi pencarian
            searchAll = boolean yang bernilai true jika ingin mencari lebih dari 1 file,
                        false jika hanya mencari 1 file
            graph = Graf yang digunakan untuk visualisasi
            timeDelay = jeda waktu antara progres pembuatan graf

  Keluaran : Seluruh lokasi file yang dicari dalam bentuk larik string
}

Deklarasi
    res : list of string

Algoritma
    recusivelySearchDFS(fileToSearch, path, searchAll, ref graph, timeDelay, path)
    res = copyList(Global.result)
    Global.clean()
    return res



procedure recursivelySearchDFS(fileToSearch, path : string, searchAll: boolean, graph :
Graph, timeDelay : int, pathBapak : string, pathNode : string)
```

```
{ Melakukan proses pencarian DFS secara rekursif.

  Masukan : fileToSearch = nama file yang akan dicari
            path = lokasi pencarian
            searchAll = boolean yang bernilai true jika ingin mencari lebih dari 1 file,
                        false jika hanya mencari 1 file
            graph = Graf yang digunakan untuk visualisasi
            timeDelay = jeda waktu antara progres pembuatan graf
            pathBapak = lokasi bapak dari node
            pathNode = lokasi node

  Initial State : Seluruh masukan terdefinisi
  Final State : Menambahkan path hasil ke Global.result kemudian memanggil
                kembali fungsi ini dengan path baru merupakan folder dari path sekarang
                jika masih ada
}

Deklarasi
    files : list of string
    dirs : list of string
    parent, child : string
Algoritma
    files <- getFiles(path)                  { Mengambil semua nama file dalam path }
    foreach file in files do
        parent <- pathNode.Last()
        child <- file.Last()

        { Menambahkan node child }
        Global.addNode(child)

        { Menambahkan node parent jika belum ada }
        Microsoft.Msagl.Drawing.Node parentNode = NIL
        if(graph.FindNode(parent) = NIL) then
            parentNode <- graph.AddNode(parent)
            Global.addNode(parent)
        end if
        else
            parentNode = graph.FindNode(parent)
        end else

        if (file = fileToSearch) then               { Jika file ditemukan }
            Global.result <- Global.result ∪ file
            colorEdge(file, SkyBlue)   { Mengubah warna edge yang benar menjadi SkyBlue}
            if (searchAll = false and Global.isRunning) then
                Global.isRunning <- false
                return
            end if
        end if
        else
            colorEdge(file, Orange)     { Mengubah warna edge yang salah menjadi Orange}
        end else
        Global.updateGraph(timeDelay)      { Mengubah tampilan graf sesuai time delay }

    end foreach

    dirs <- getDirs(path)                   { Mengambil semua nama folder dalam path }
    foreach dir in dirs do
            if (Global.isRunning) then      { Mengecek apakah program masih berjalan }
            parent <- pathNode.Last()
            child <- file.Last()

            { Menambahkan node child }
            Global.addNode(child)

            { Menambahkan node parent jika belum ada }
```

```
            Microsoft.Msagl.Drawing.Node parentNode = NIL
            if(graph.FindNode(parent) = NIL) then
                parentNode <- graph.AddNode(parent)
                Global.addNode(parent)
            end if
            else
                parentNode = graph.FindNode(parent)
            end else
            Global.updateGraph(timeDelay)   { Mengubah tampilan graf sesuai time delay }

            { Memanggil fungsi ini lagi untuk mencari file dalam folder anak }
            recursivelySearchDFS(fileToSearch, path + dir, searchAll, graph, timeDelay,
pathBapak, pathNode + child)
        end if
    end foreach
```

d. Algoritma BFS

```
function searchBFS(fileToSearch, pathToSearch : string, searchAll : boolean, graph :
Graph, timeDelay : int) -> list of string
{ Melakukan inisiasi pencarian DFS. Global.result adalah larik string yang merupakan
variabel Global
  (bisa diakses di mana saja).

  Masukan : fileToSearch = nama file yang akan dicari
            path = lokasi pencarian
            searchAll = boolean yang bernilai true jika ingin mencari lebih dari 1 file,
                        false jika hanya mencari 1 file
            graph = Graf yang digunakan untuk visualisasi
            timeDelay = jeda waktu antara progres pembuatan graf

  Keluaran : Seluruh lokasi file yang dicari dalam bentuk larik string

}

Deklarasi
    res : list of string
    files : list of string
    dirs : list of string
    parent, child : string

Algoritma
    Global.pathQueue.Enqueue(pathToSearch)     { Menambahkan path awal ke antrian }

    while (Global.pathQueue.count > 0) do      { selagi masih ada path di antrian }
        path = Global.pathQueue.Dequeue()      { Mengambil path dari antrian pertama }
        files <- getFiles(path)                { Mengambil semua nama file dalam path }
        foreach file in files do
            parent <- pathNode.Last()
            child <- file.Last()

            { Menambahkan node child }
            Global.addNode(child)

            { Menambahkan node parent jika belum ada }
            Microsoft.Msagl.Drawing.Node parentNode = NIL
            if(graph.FindNode(parent) = NIL) then
                parentNode <- graph.AddNode(parent)
                Global.addNode(parent)
            end if
            else
                parentNode = graph.FindNode(parent)
            end else

            if (file = fileToSearch) then              { Jika file ditemukan }
```

```
                Global.result <- Global.result ∪ file
                colorEdge(file, SkyBlue){Mengubah warna edge yang benar menjadi SkyBlue}
                if (searchAll = false) then
                    res = copyList(Global.result);
                    Global.clean();
                    return res
                end if
            end if
            else
                colorEdge(file, Orange) { Mengubah warna edge yang salah menjadi Orange}
            end else
            Global.updateGraph(timeDelay)   { Mengubah tampilan graf sesuai time delay }

        end foreach

        dirs <- getDirs(path)                 { Mengambil semua nama folder dalam path }
        foreach dir in dirs do
                parent <- pathNode.Last()
                child <- file.Last()

                { Menambahkan node child }
                Global.addNode(child)

                { Menambahkan node parent jika belum ada }
                Microsoft.Msagl.Drawing.Node parentNode = NIL
                if(graph.FindNode(parent) = NIL) then
                    parentNode <- graph.AddNode(parent)
                    Global.addNode(parent)
                end if
                else
                    parentNode = graph.FindNode(parent)
                end else

                Global.pathQueue.Enqueue(dir)     { Memasukkan dir ke dalam antrian }
                Global.updateGraph(timeDelay) {Mengubah tampilan graf sesuai time delay}
            end if
        end foreach
    end while

    res = utility.copyList(Global.result)
    Global.clean()
    return res
```

2. Penjelasan Struktur Data dan Spesifikasi Program

Program ini menggunakan 4 struktur data yaitu Queue, Array, Hash Table, dan Tree.

a. Queue

Queue digunakan pada metode pencarian BFS. Queue yang digunakan disini adalah queue sederhana untuk menyimpan urutan path yang akan di proses.

b. Array

Array digunakan untuk menyimpan hasil path yang ditemukan.

c. Hash Table

Hash Table digunakan untuk menyimpan edge yang telah dibuat berdasarkan id dari edge yang telah ditetapkan. Hash Table juga digunakan untuk menyimpan serta mengenerate id node baru. Pertama akan diperiksa id node yang akan ditambahkan ke tree, jika node sudah ada akan diberi tambahan " (N)" dimana N merupakan jumlah node dengan id yang sama pada tree.

d. Tree

Tree digunakan untuk menggambarkan path pencarian. Setiap node merepresentasikan file atau folder dan setiap edge merepresentasikan hubungan parent dan child dimana arah yang ditunjuk merupakan child.

Spesifikasi Program:

Spesifikasi GUI:

1. Program dapat menerima input folder dan query nama file.
2. Program dapat memilih untuk menampilkan satu hasil saja atau menemukan semua file yang memiliki nama file sama persis dengan input query
3. Program dapat memilih algoritma yang digunakan.
4. Program dapat menampilkan pohon hasil pencarian file tersebut dengan memberikan keterangan folder/file yang sudah diperiksa, folder/file yang sudah masuk antrian tapi belum diperiksa, dan rute folder serta file yang merupakan rute hasil pertemuan.
5. **(Bonus)** Program dapat menampilkan progress pembentukan pohon dengan menambahkan node/simpul sesuai dengan pemeriksaan folder/file yang sedang berlangsung.
6. Program dapat menampilkan hasil pencarian berupa rute/path (bisa lebih dari satu jika memilih menemukan semua file) serta durasi waktu algoritma.
7. GUI dapat dibuat **sekreatif** mungkin asalkan memuat 5(6 jika mengerjakan bonus) spesifikasi di atas.

Program yang dibuat harus memenuhi **spesifikasi wajib** sebagai berikut:

1. Buatlah program dalam bahasa C# untuk melakukan penelusuran Folder Crawling sehingga diperoleh hasil pencarian file yang diinginkan. Penelusuran harus memanfaatkan algoritma BFS dan DFS.
2. Awalnya program menerima sebuah input folder pada direktori yang ada dan nama file yang akan dicari oleh program.
3. Terdapat dua pilihan pencarian, yaitu:
    a. Mencari 1 file saja
    Program akan memberhentikan pencarian ketika sudah menemukan file yang memiliki nama sama persis dengan input nama file.
    b. Mencari semua kemunculan file pada folder root
    Program akan berhenti ketika sudah memeriksa semua file yang terdapat pada folder root dan program akan menampilkan daftar semua rute file yang memiliki nama sama persis dengan input nama file
4. Program kemudian dapat menampilkan visualisasi pohon pencarian file berdasarkan informasi direktori dari folder yang di-input. Pohon hasil pencarian file ini memiliki root adalah folder yang di-input dan setiap daunnya adalah file yang ada di folder root tersebut. Setiap folder/file direpresentasikan sebagai sebuah node atau simpul pada pohon. Cabang pada pohon menggambarkan folder/file yang terdapat di folder parent-nya.
    Visualisasi pohon juga harus disertai dengan keterangan node yang sudah diperiksa, node yang sudah masuk antrian tapi belum diperiksa, dan node yang bagian dari rute hasil penemuan.
    Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk melakukan visualisasi adalah MSAGL (https://github.com/microsoft/automatic-graph-layout) Berikut ini adalah panduan singkat terkait penggunaan MSAGL oleh tim asisten yang dapat diakses pada:

https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVl3MY6gt1t-PL30LA/edit?usp=sharing

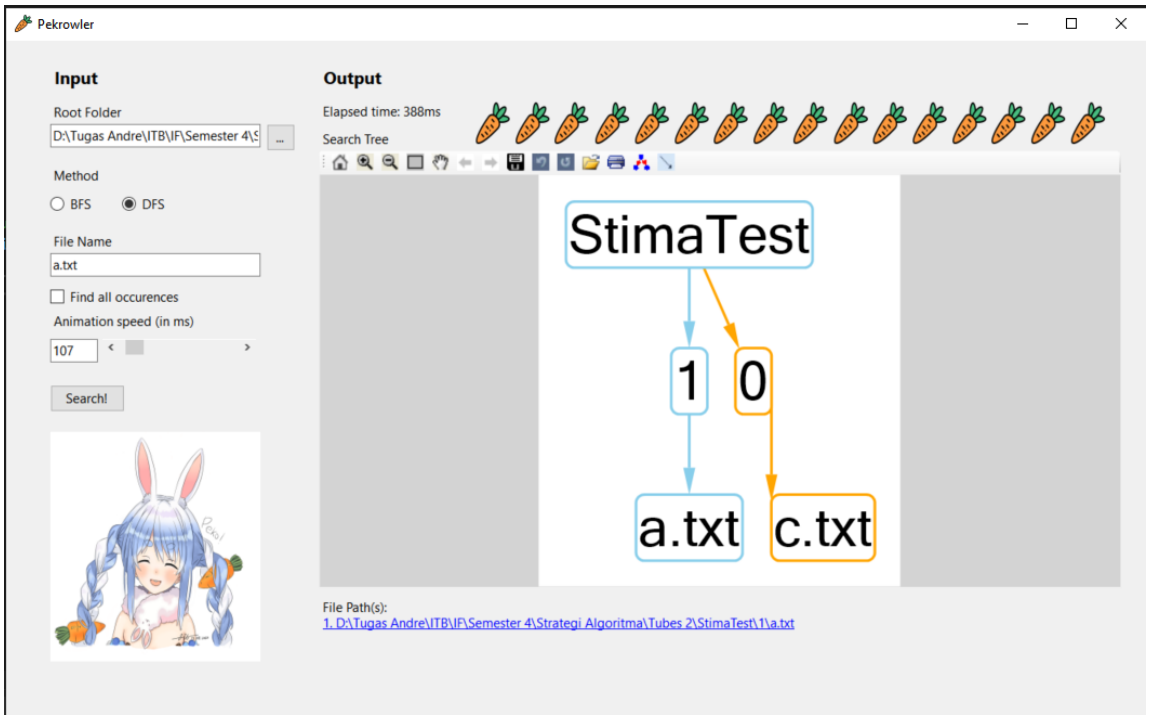
5. Program juga dapat menyediakan hyperlink pada setiap hasil rute yang ditemukan. Hyperlink ini akan membuka folder parent dari file yang ditemukan. Folder hasil hyperlink dapat dibuka dengan browser atau file explorer.
6. Mahasiswa tidak diperkenankan untuk melihat atau menyalin library lain yang mungkin tersedia bebas terkait dengan pemanfaatan BFS dan DFS. Tapi untuk algoritma lainnya seperti string matching dan akses directory, diperbolehkan menggunakan library jika ada.

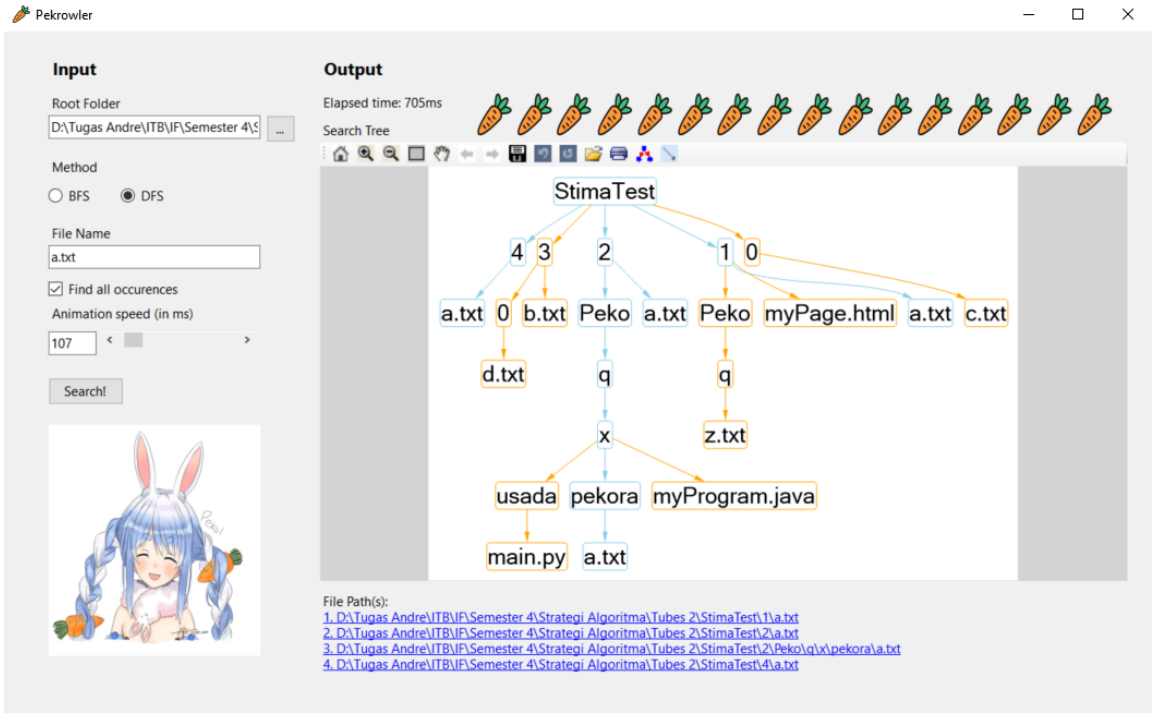
3. Tata Cara Penggunaan

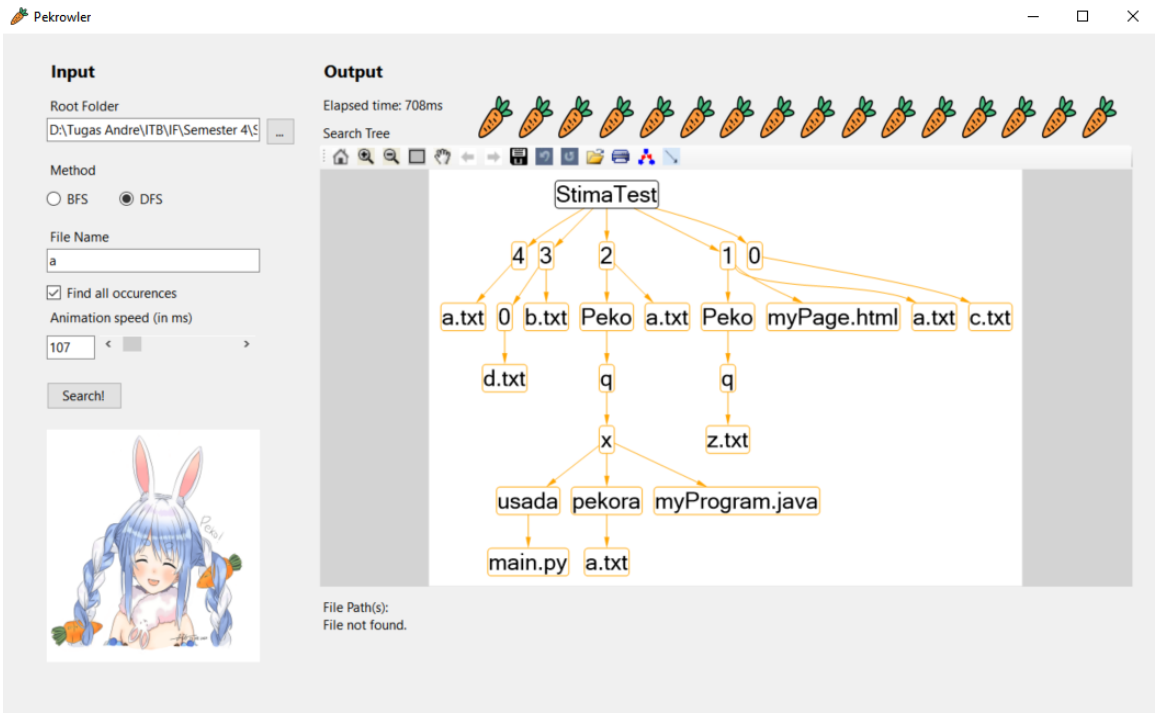
Interface program memiliki 5 input yaitu root folder, metode pencarian yang digunakan, nama file yang dicari, pilihan untuk mencari semua kemunculan file, serta delay animasi yang diinginkan. Sebelum bisa menekan tombol search, pengguna wajib mengisi root folder, metode, nama file, serta delay animasi. Output yang dikeluarkan berupa hyperlink untuk membuka folder yang terdapat di bawah graf folder.

4. Hasil Pengujian

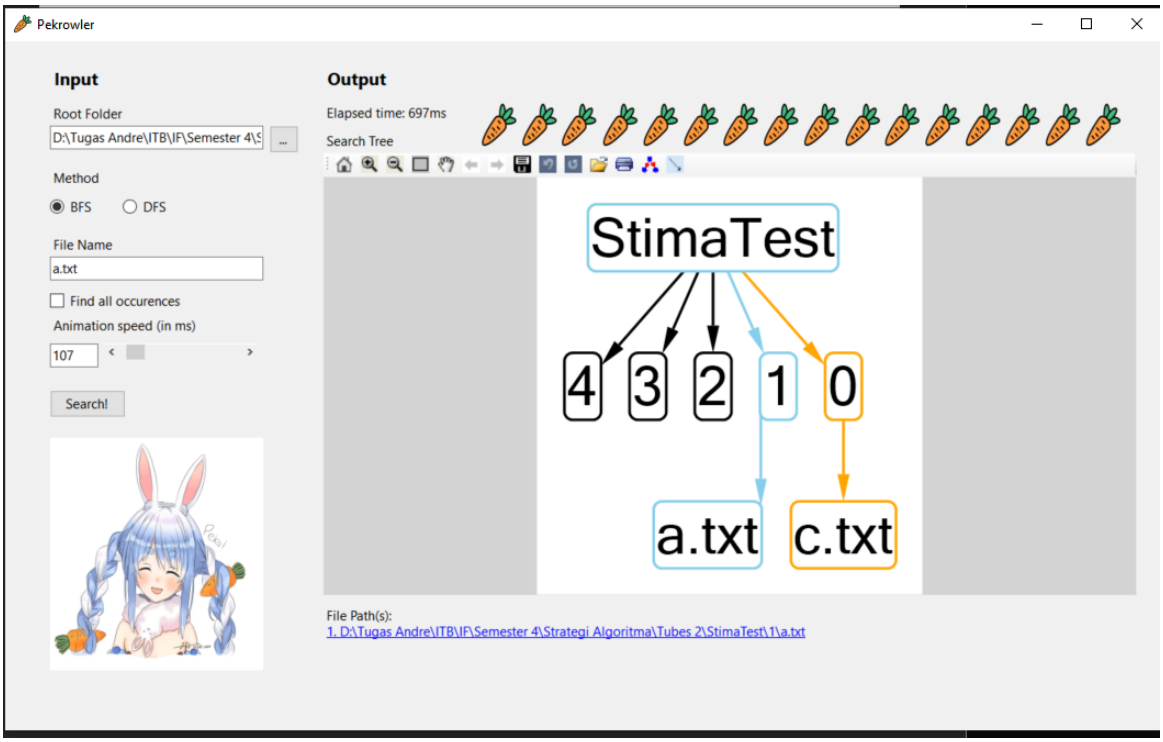
pengujian dengan DFS:

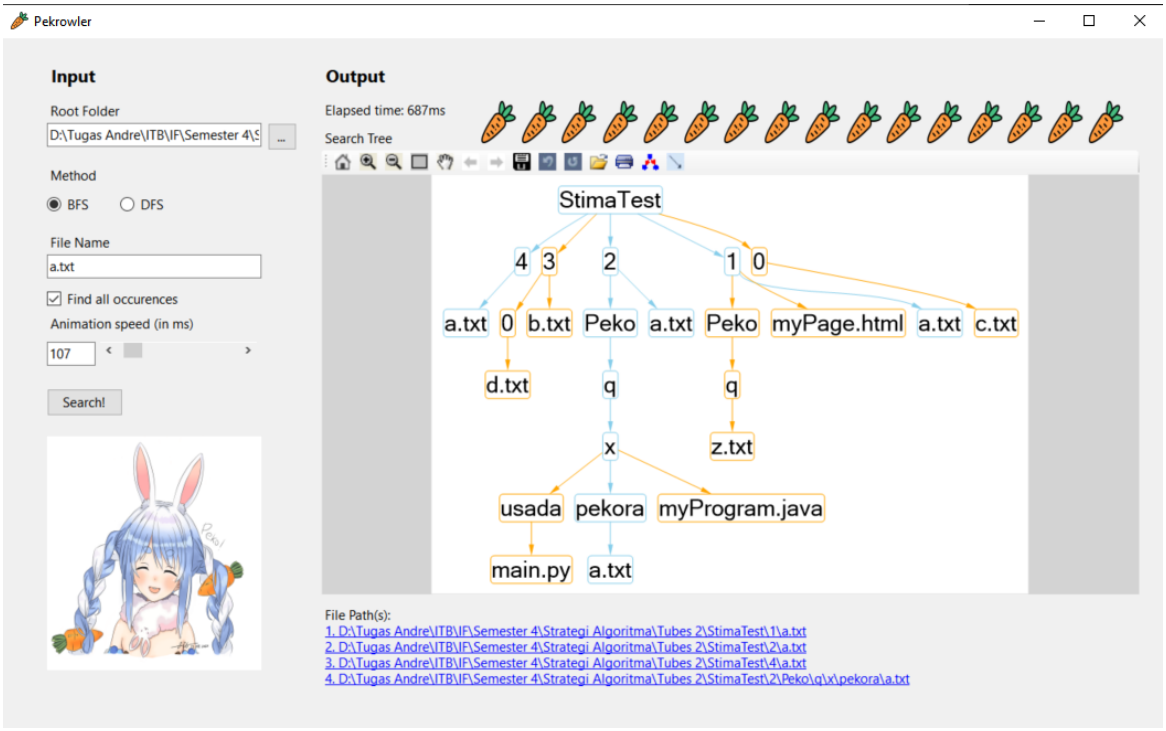


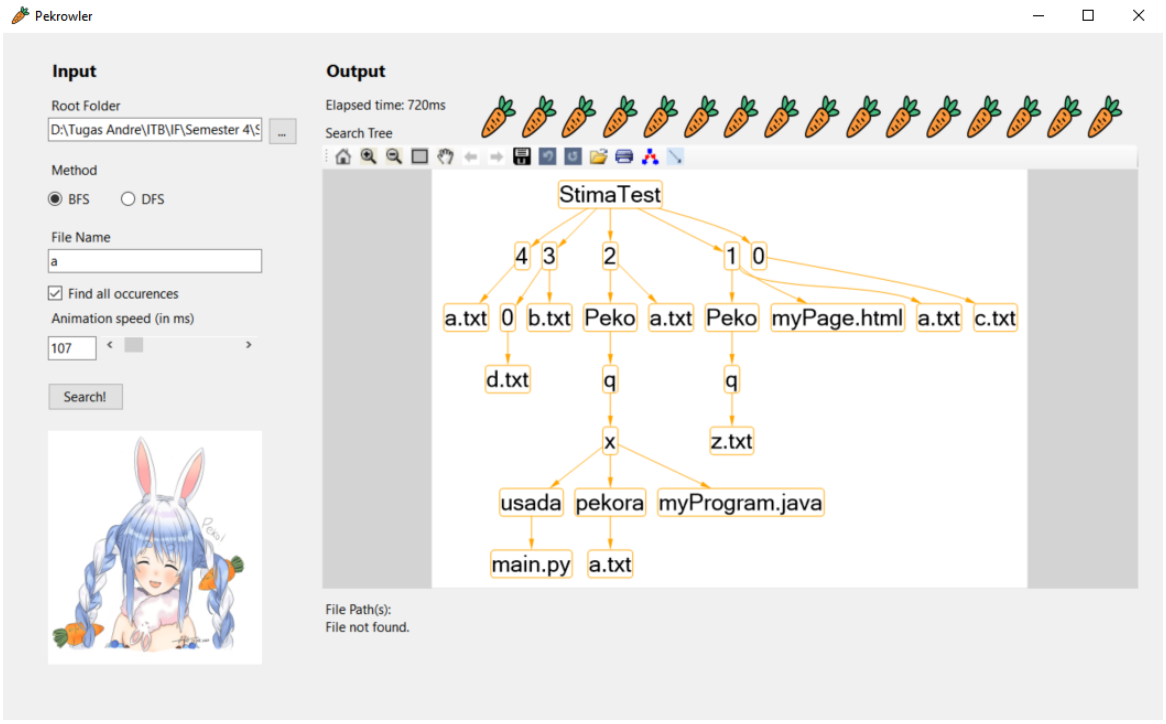




Pengujian dengan BFS:







5. Analisis Desain Solusi Algoritma BFS dan DFS

Pada hasil pengujian di atas, dilakukan pencarian file a.txt pada root folder test stima. pada hasil pencarian file pertama terlihat bahwa a.txt ditemukan hanya dengan 4 pencarian(1 pencarian dilambangkan oleh 1 panah) pada metode DFS sedangkan a.txt baru ditemukan pada pencarian ke-6 pada metode BFS. Hal ini disebabkan karena metode DFS akan melakukan pencarian ke bawah terlebih dahulu hingga tidak ditemukan folder lagi. Sedangkan BFS akan melakukan pencarian pada seluruh file dan folder pada satu parent terlebih dahulu sebelum melanjutkan ke folder yang lebih bawah.

Algoritma BFS akan lebih efektif jika file berada pada kedalaman yang rendah dan pada satu parent folder tidak memiliki banyak folder child. Sedangkan untuk algoritma DFS, pencarian akan lebih efektif jika path file berada pada folder yang pertama dicari atau kedalaman dari setiap folder dangkal dan setiap parent memiliki banyak folder child.

Jika dilihat dari hasil pengujian di atas, a.txt ditemukan lebih cepat menggunakan DFS karena a.txt berada pada folder ke-2 dan folder ke-1 pathnya dangkal. Namun ketika dilakukan pencarian a.txt yang paling kiri di pohon, algoritma BFS menemukan a.txt lebih cepat karena tidak terlalu dalam serta algoritma DFS terhambat di folder 2 dan 1 yang cukup dalam.

BAB 5
KESIMPULAN DAN SARAN

1. Kesimpulan

Algoritma traversal graf dengan BFS dan DFS dapat memecahkan persoalan *file search* dengan cukup efektif. Representasi file menjadi suatu struktur graf menghasilkan pencarian yang sistematis. Implementasi kami dengan bahasa pemrograman C# untuk persoalan ini juga diimplementasikan dengan baik tanpa masalah. GUI yang diimplementasikan menggunakan .NET framework dan Windows Form pun fungsional dan memiliki fitur lengkap, seperti animasi, yang menggambarkan proses traversal graf dengan jelas. Dengan ini, aplikasi file search yang kami buat memenuhi semua spesifikasi yang diberikan.

2. Saran

Saran yang dapat kami berikan berdasarkan pengalaman dan hasil yang didapat dari pengerjaan tugas besar ini adalah:

1. Setiap node dapat dibedakan dari id masing-masing node
2. Delay time yang terlalu kecil saat menggunakan animasi dapat berpotensi membuat kode error
3. Jika button program tidak dilock ketika melakukan pencarian dapat berpotensi terjadi error
4. GUI dapat dihias agar tidak terkesan monoton dengan font default

Link Github : https://github.com/IMYELI/Pekrowler.git

Link Youtube : https://youtu.be/2pGakMDzdv0