Laporan Tugas Besar IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS dalam implementasi *Folder Crawling*

Disusun oleh

Samuel Christopher - 13520075

Patrick Amadeus Irawan - 13520109

Azmi Alfatih Shalahuddin - 13510158

Teknik Informatika

Institut Teknologi Bandung

Bandung

2022

BAB 1

DESKRIPSI TUGAS

Dalam tugas besar ini, kami 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), kami dapat menelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang Anda inginkan. Kami juga diminta untuk memvisualisasikan hasil dari pencarian folder tersebut dalam bentuk pohon.

BAB 2

LANDASAN TEORI

1. Graf

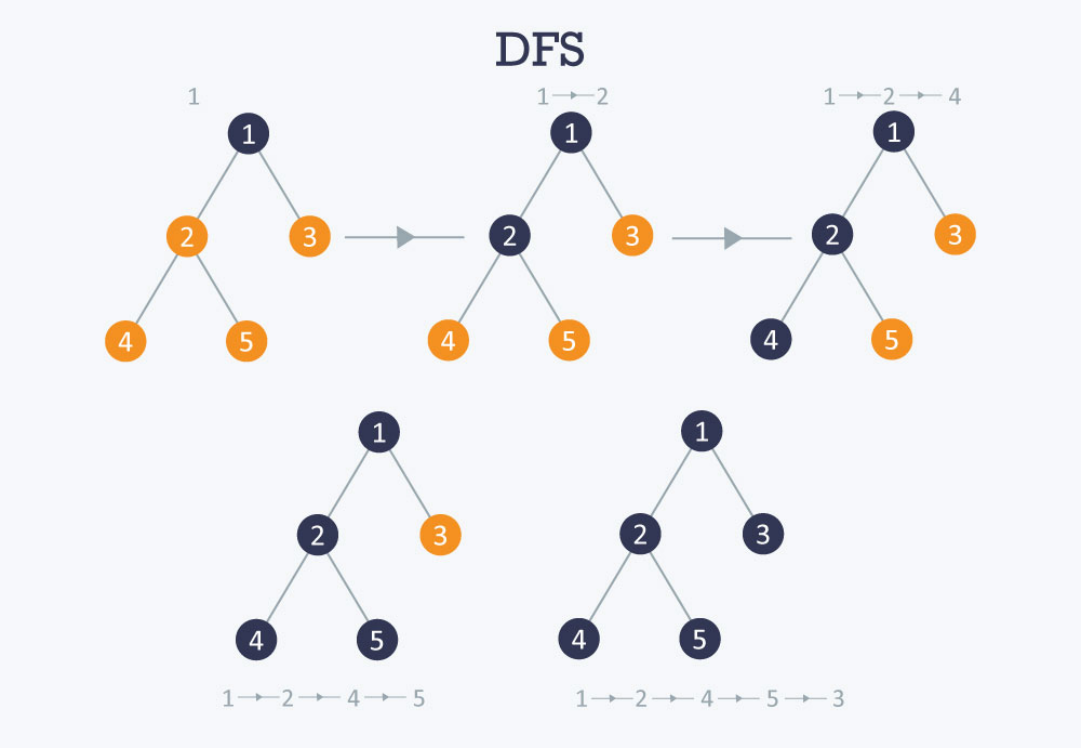
   Graf merupakan sekumpulan objek atau data terstruktur di mana beberapa pasangan objek mempunyai hubungan ataupun keterkaitan tertentu. Secara informal, suatu graf adalah himpunan benda-benda yang disebut "simpul" (vertex atau node) yang terhubung oleh "sisi" (edge) atau "busur" (arc). Biasanya graf digambarkan sebagai kumpulan titik-titik (melambangkan "simpul") yang dihubungkan oleh garis-garis (melambangkan "sisi") atau garis berpanah (melambangkan "busur"). Suatu sisi dapat menghubungkan suatu simpul dengan simpul yang sama. Sisi yang demikian dinamakan "gelang" (loop).

   Banyak sekali struktur yang bisa direpresentasikan dengan graf, dan banyak masalah yang bisa diselesaikan dengan bantuan graf. Jaringan persahabatan pada Facebook bisa direpresentasikan dengan graf, yakni simpul-simpulnya adalah para pengguna Facebook dan ada sisi antar pengguna jika dan hanya jika mereka berteman. Perkembangan algoritma untuk menangani graf akan berdampak besar bagi ilmu komputer.

2. DFS

   DFS (*Depth First Search)* merupakan algoritma pencarian graf, untuk menemukan simpul tertentu (node). Langkah-langkah dari DFS adalah:
   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 di mana semua simpul yang bertetanggaan sudah dikunjungi, maka akan terjadi pencarian runut-balik (*backtracking)* ke simpul terakhir yang dikunjungi oleh u sebelumnya dan mempunyai simpul v yang belum dikunjungi
   5. Pencarian berakhir bila sudah menemukan simpul tertentu, atau tidak ada lagi simpul yang belum dikunjungi dan dapat dicapai dari simpul yang telah dikunjungi

   Berikut ilustrasi DFS:



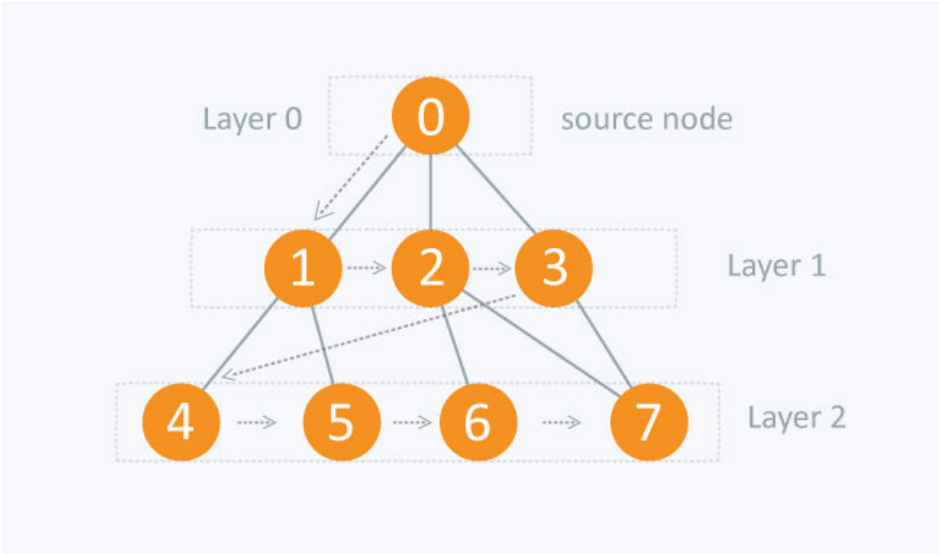
Source: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/

3. BFS

BFS (*Breadth First Search)* merupakan algoritma pencarian graf, untuk menemukan simpul tertentu (node). Langkah-langkah dari BFS adalah:

1. Kunjungi simpul v
2. Kunjungi seluruh simpul yang bertetangga dengan simpul v
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul yang dikunjungi

Berikut ilustrasi DFS:



4. Penjelasan singkat mengenai C# desktop application development

GUI ini dibuat dengan bahasa pemrograman C# di dalam aplikasi visual studio, dengan menggunakan windows Form. Windows Form atau sering disebut dengan WinForm adalah Class library buatan microsoft yang bersifat GUI atau Graphical User Interface. Windows Form tergabung dalam framework .NET. Tujuan diciptakan Windows Form adalah untuk mempermudah developer dalam membuat suatu aplikasi berbasis desktop.

BAB 3

ANALISIS PEMECAHAN MASALAH

1. Langkah Pemecahan Masalah
   Langkah yang kami gunakan dalam pemecahan masalah ini adalah membuat struktur data berupa *map* untuk *map* keadaan sebuah simpul. Keadaan simpul dibagi menjadi 3 yaitu simpul yang sudah dikunjungi tetapi tidak ada simpul yang dibutuhkan, simpul yang sudah dikunjungi dan ada simpul yang dibutuhkan, dan simpul yang sudah dikunjungi namun tidak dikunjungi lebih lanjut karena sudah ditemukan simpul tertentu di pencarian yang sedang dilakukan

2. Proses mapping persoalan menjadi elemen-elemen algoritma BFS dan DFS

   a. BFS

   Dalam algoritma pencarian simpul(file) menggunakan BFS, kami menggunakan pencarian iteratif dimulai dari simpul pertama yaitu root dari folder yang diberikan oleh user. Kami mengambil head folder, dan mulai melakukan iterasi pada semua file yang ada di folder root, lalu jika ditemukan file yang sesuai dengan yang dicari, maka akan berhenti iterasi, lalu mengembalikan graf simpul yang sudah dikunjungi. Jika diiterasi folder root pertama tidak ditemukan file yang diinginkan, maka akan mencari seluruh folder di dalam root folder dan dinamakan *subEntryDirectories*, dan akan memulai iterasi dari awal. Hal ini dilakukan sampai tidak ada lagi *subEntryDirectories*. Kasus yang terjadi dapat terbagi menjadi dua yaitu jika user ingin menemukan seluruh file atau hanya ingin menemukan keberadaan pertama file tersebut.

   b. DFS

   Dalam algoritma pencarian simpul(file) menggunakan DFS, kami menggunakan pencarian rekursif dimulai dari simpul pertama yaitu root dari folder yang diberikan oleh user. Kami mengambil head folder, lalu melakukan iterasi pada semua file yang terdapat pada root folder. Jika ditemukan, maka akan langsung dikembalikan, dan jika tidak, maka akan memulai rekursi untuk folder yang berada di dalam folder root. Rekursi akan berhenti jika sudah tidak ada lagi folder yang ada di dalam folder atau tidak ada file yang ada di dalam folder. Kasus yang terjadi dapat terbagi menjadi dua yaitu jika user ingin menemukan seluruh file atau hanya ingin menemukan keberadaan pertama file tersebut.

3. Contoh Ilustrasi Kasus
    a. Kasus pencarian file dengan metode BFS
    Pemilihan kasus ini dilakukan dengan memilih radio button BFS dengan checkbox All Occurence dibiarkan tidak dicentang. Pemilihan kasus ini akan menghasilkan visualisasi tree hasil algoritma Breadth First Search, dimana garis merah menunjukkan folder yang tidak mengandung file yang dicari , garis hitam, menunjukkan folder yang sudah dikunjungi dan masih memiliki subdirektori tetapi kondisi file sudah ditemukan di tempat lain, sedangkan garis hijau menunjukkan jalur pencarian dimana file ditemukan.
    b. Kasus pencarian file dengan metode DFS
    Pemilihan kasus ini dilakukan dengan memilih radio button DFS dengan checkbox All Occurence dibiarkan tidak dicentang. Pemilihan kasus ini akan menghasilkan visualisasi tree hasil algoritma Breadth First Search, dimana garis merah menunjukkan folder yang tidak mengandung file yang dicari, garis hijau menunjukkan jalur pencarian dimana file ditemukan.
    c. Kasus pencarian semua kemungkinan file dengan metode BFS
    Pemilihan kasus ini dilakukan dengan memilih radio button BFS dengan checkbox All Occurence dibiarkan tidak dicentang. Pemilihan kasus ini akan menghasilkan visualisasi tree hasil algoritma Breadth First Search, dimana garis merah menunjukkan folder yang tidak mengandung file yang dicari , garis hijau menunjukkan jalur pencarian dimana file ditemukan. Pencarian akan dilakukan hingga semua kemungkinan direktori dicapai
    d. Kasus pencarian semua kemungkinan file dengan metode DFS
    Pemilihan kasus ini dilakukan dengan memilih radio button BFS dengan checkbox All Occurence dibiarkan tidak dicentang. Pemilihan kasus ini akan menghasilkan visualisasi tree hasil algoritma Breadth First Search, dimana garis merah menunjukkan folder yang tidak mengandung file yang dicari , garis hijau menunjukkan jalur pencarian dimana file ditemukan. Pencarian akan dilakukan hingga semua kemungkinan direktori dicapai

BAB 4

IMPLEMENTASI DAN PENGUJIAN

1. Implementasi program (*pseudocode* program utama)

   a. Main program

      i. Form

```
class MainForm():
    procedure StartDirectory_Click(input/output DirTextInput)
        KAMUS LOKAL
            openFileDialog : CommonFileDialog
        ALGORITMA
            openFileDialog.IsFolderPicker <- true;
            if (openFileDialog.ShowDialog() = CommonFileDialogResult.Ok) then
                DirTextInput.Text <- openFileDialog.FileName;

    procedure SearchButton_Click(input/output WarningLabel : string,
    input BFSbutton, DFSbutton, AllOccurence :
    bool , FoundDirs : ListBox, GraphPanel : Panel , TimeSpentText : Label)
        KAMUS LOKAL
            g <- Graph
            ndoes <- map(string,int)
            foundPath <- array of string
            stopwatch <- Stopwatch
        ALGORITMA
            if ((not BFSbutton && not DFSbutton) || DirTextInput.Text = "" || FileInput.Text = "") then
                WarningLabel.Text <- "";
                {sleep 20 detik}
                WarningLabel.Text <- "Please fill in required fields";
            else

                WarningLabel.Text <- "";

                // Clearing Old Output(s)
                FoundDirs.Items <- nil
                GraphPanel.Controls <- nil;

                nodes <- {}
                foundPath <- []
                if (BFSbutton) then
                    {stopwatch mulai}
                    nodes <- BFSsearching(AllOccurence, DirTextInput.Text, FileInput.Text, foundPath);
                    {stopwatch berakhir}
                else
                    {stopwatch mulai}
                    nodes <- DFSsearching(AllOccurence, DirTextInput.Text, FileInput.Text, foundPath);
                    {stopwatch berakhir}

                GraphPanel.Controls.Add(ShowGraph(nodes, BFSbutton));


                for path in foundPath
                    FoundDirs <- FoundDirs + path

                TimeSpentText.Text <- "Time Spent: " + {detik proses} +"s";


    procedure FoundDirs_SelectedIndexChanged(object sender, EventArgs e)
        KAMUS LOKAL
            -
        ALGORITMA
            {launch file explorer sesuai dengan pilihan file}


```

ii. Graph Maker

```
function ShowGraph(nodes : map(string,int) , type : boolean)
KAMUS LOKAL
    view <- Microsoft.Msagl.Drawing.Viewer
    graph <- Microsoft.Msagl.Drawing.Graph
    pathParsed <- array of string
    keyParent <- string
ALGORITMA
    view <- nil
    graph <- nil
    for k,v in nodes.Reverse()
        pathParsed <- {Split k dengan delimiter \}
        pathParsed.RemoveAt(pathParsed.Count - 1);

        keyParent = {Join pathParsed dengan delimiter \}

        if v == 1 then
            graph <- graph + {Edge baru antara keyParent <-> k dengan warna hijau}
        else if v == 0 then
            if (not type)
                graph <- graph + {Edge baru antara keyParent <-> k dengan warna merah}
            else
                graph <- graph + {Edge baru antara keyParent <-> k dengan warna hitam}
        else
            graph <- graph + {Edge baru antara keyParent <-> k dengan warna merah}

        graph.text <- jadikan direktori akhir
        graph.color <- jadikan warna kuning pudar

    view.Graph <- graph;
    view.Dock <- System.Windows.Forms.DockStyle.Fill;
    view.BackgroundColor <- White;

    -> view
```

b. BFS

```
function BFSsearching(type: boolean, root: string, filename: string) -> map(string, int)
KAMUS LOKAL
    result : map(string, int)
    queue : queue(string)
    subResult : map(string, int)
    found : boolean
    entryFiles : array of string
    subEntryDirectories : array of string
    current : string
ALGORITMA
    queue.Add(root)
    found <- false
    while (!queue.IsEmpty)
        current <- queue.Dequeue()
        entryFiles <- GetAllFilesIn(current)
        for (entryFile in entryFiles)
            if (entryFile == filename)
                result.Add(entryFIle, 1)
                found <- true
                MakeColorParent(entryFile, result, true)
            else
                result.Add(entryFIle, -1)
                MakeColorParent(entryFile, result, false)

        { cek apakah tipe bfs adalah first occurence atau tidak }
        if(!type)
            { jika sudah ditemukan file yang memenuhi }
            if (found)
                return result

        subEntryDirectories <- GetAllDirectoriesIn(current)
        for (subEntryDirectory in subEntryDirectories)
            result.Add(subEntryDirectory, 0)
            queue.Add(subEntryDirectory)


    return result

```

c. DFS

```
function DFSsearching(type: boolean, root: string, filename: string) -> map(string, int)
KAMUS LOKAL
    result : map(string, int)
    subResult : map(string, int)
    found : boolean
    entryFiles : array of string
    subEntryDirectories : array of string

ALGORITMA
    entryFiles <- GetAllFilesIn(root)
    found <- false
    for (entryFile in entryFiles)
        if (entryFile == filename)
            result.Add(filename, 1)
            found <- true

        { cek apakah tipe dfs adalah first occurence atau tidak }
        if(!type)
            { jika sudah ditemukan file yang memenuhi }
            if (found)
                return result

    subEntryDirectories <- GetAllDirectoriesIn(root)
    for (subEntryDirectory in subEntryDirectories)
        subResult <- DFSsearching(type, subEntryDirectory, filename)

        { menambahkan semua subResult ke result }
        for (key in subResult.Keys)
            result.Add(key, subResult[key])
            { menandai 1 untuk setiap parent yang mempunyai filepath result[key] }
            if (result[key] == 1)
                MakeColorParent(key, result, true)

        { cek apakah tipe dfs adalah first occurence atau tidak }
        if(!type)
            for (key in subResult.Keys)
                { jika sudah ditemukan file yang memenuhi }
                if (subResult[key] == 1)
                    return result

    return result

```

2. Penjelasan struktur data

   Struktur data yang kami gunakan untuk visualisasi graf adalah hashmap, yang isinya berupa string dan integer. String dalam map merupakan string dari *filepath* yang dikunjungi oleh program, dan integer merupakan keadaan dari *filepath* yang sedang dikunjungi. Keadaan *filepath* terbagi menjadi 3 yaitu:

   1. *Filepath* yang sudah dikunjungi, namun tidak terdapat file yang diinginkan
   2. *Filepath* yang sudah dikunjungi dan terdapat file yang diinginkan
   3. *Filepath* yang sudah dikunjungi, namun tidak ditelusuri lebih dalam, karena sudah ditemukan *filepath* lain yang sudah menemukan file yang diinginkan

   Keadaan di atas digambarkan di dalam map berupa integer dengan value -1,1,0 terurut sesuai deskripsi di atas

3. Penjelasan tata cara penggunaan program

   Navigasi ke folder bin dan jalankan Dashboard.exe (perhatikan, pastikan tipe file berupa Application / exe, terdapat file config dengan nama serupa). Jalankan aplikasi ini pada OS Windows. Akan muncul *Graphical User Interface* (GUI) sederhana dengan permintaan input yang wajib diisi oleh pengguna, yakni:

   - Starting Directory : folder start pencarian
   - File Input Text : nama file yang hendak dicari (lengkap dengan ekstensinya)
   - Check Box All Occurence : pemilihan untuk mencari semua kemungkinan atau tidak
   - Algoritma yang dipilih (BFS/DFS)

   Akan dihasilkan 3 jenis keluaran utama sebagai berikut :

   - Hyperlink : Berupa tautan menuju file yang berhasil dicari
   - Visualisasi : pohon representasi graf hasil proses pencarian file
   - Waktu Proses : waktu keberjalanan algoritma utama pencarian
   - (opsional) Warning Text : validasi apabila user input tidak lengkap

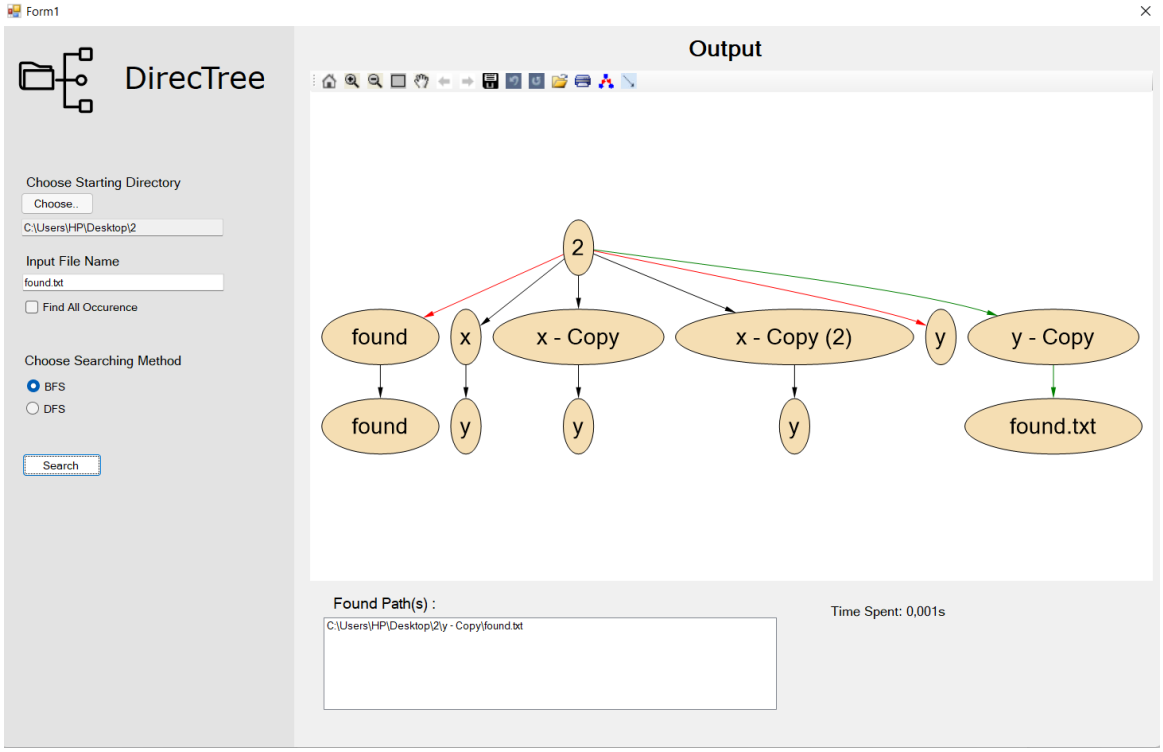
4. Hasil pengujian

TC 1 : mencari found.txt pada folder bernama “2”

Folder (non-file) tree sebagai berikut

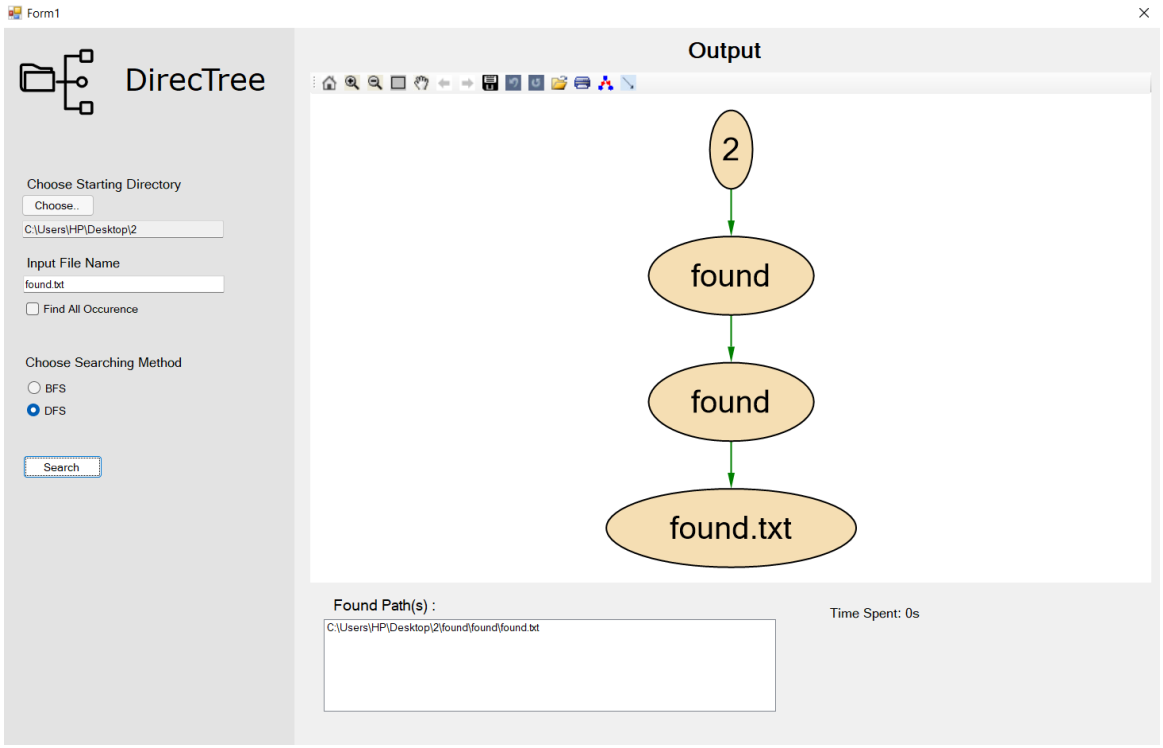
```
2:.
├───found
│   └───found
├───x
│   └───y
├───x - Copy
│   └───y
├───x - Copy (2)
│   └───y
├───y
└───y - Copy
```

BFS

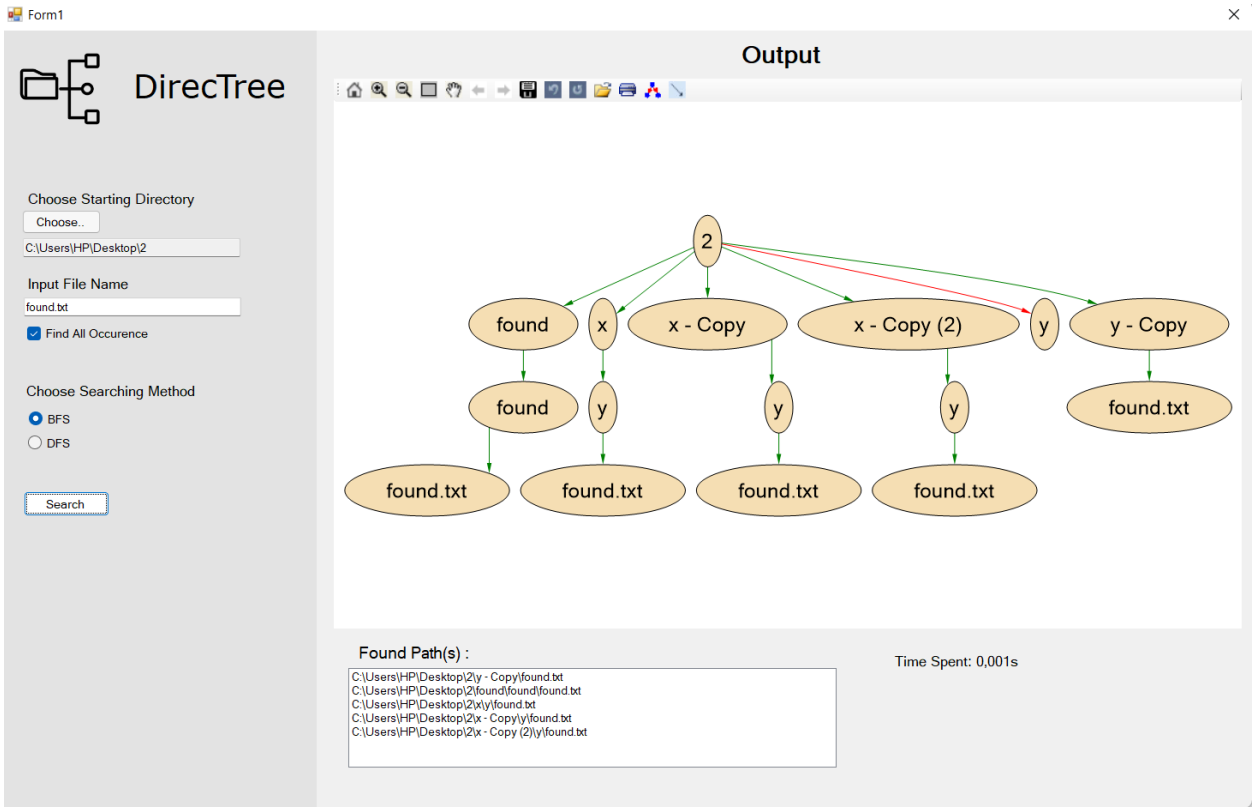


NB : algoritma kami melakukan pengecekan file terlebih dahulu sebelum melanjutkan searching, sehingga contoh garis merah pada bagian paling kiri merupakan hal yang mungkin terjadi.

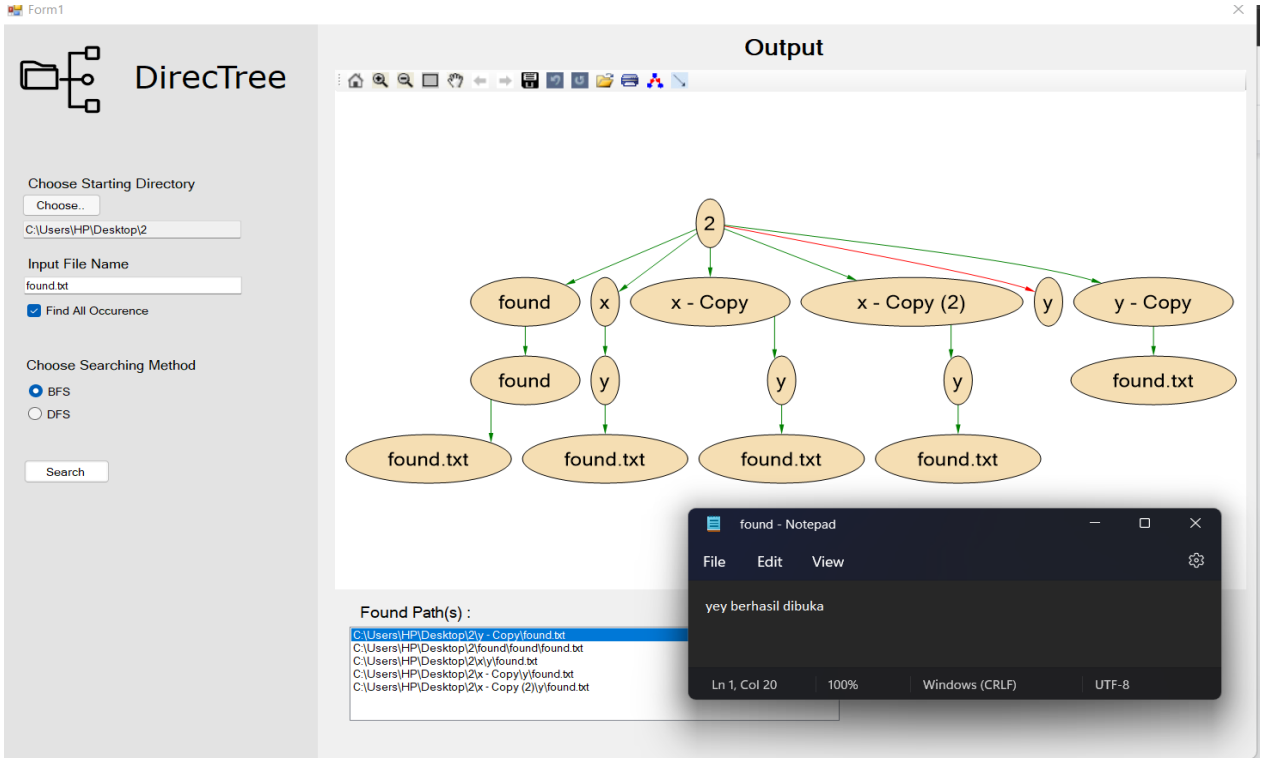
DFS


Form1
DirecTree
Choose Starting Directory
Choose..
C:\Users\HP\Desktop\2
Input File Name
found.txt
Find All Occurence
Choose Searching Method
BFS
DFS
Search
Output
2
found
found
found.txt
Found Path(s) :
C:\Users\HP\Desktop\2\found\found\found.txt
Time Spent: 0s

All Occurence BFS



Hyperlink test

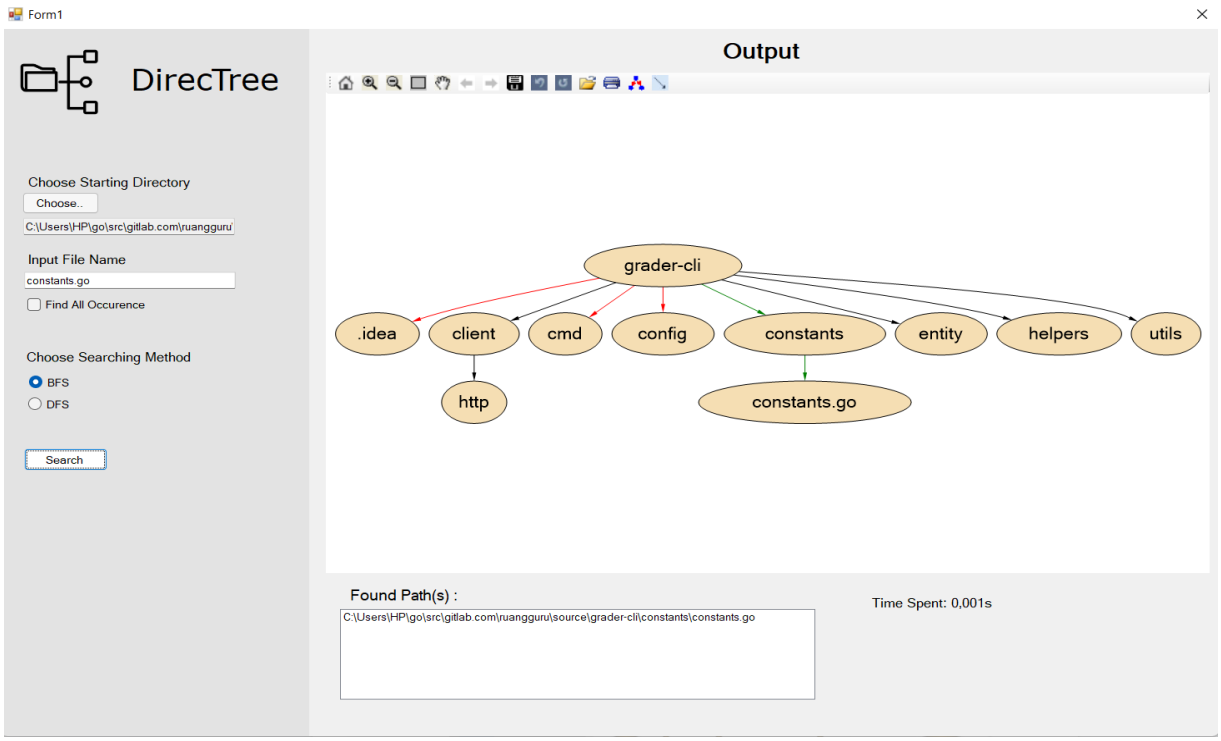


TC 2 : mencari constants.go pada folder grader-cli

Folder (non-file) tree sebagai berikut

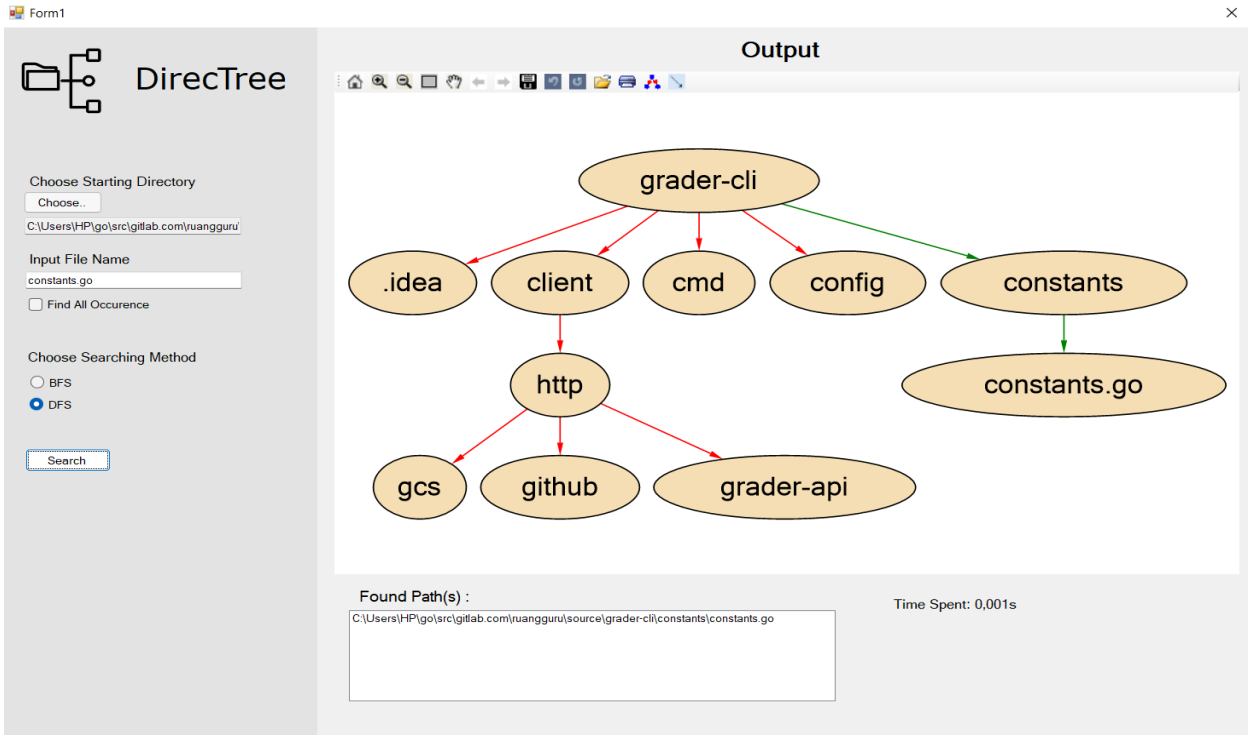
```
grader-cli:.
├───.idea
├───client
│   └───http
│       ├───gcs
│       ├───github
│       └───grader-api
├───cmd
├───config
├───constants
├───entity
├───helpers
├───utils
└───graderconf
```

BFS

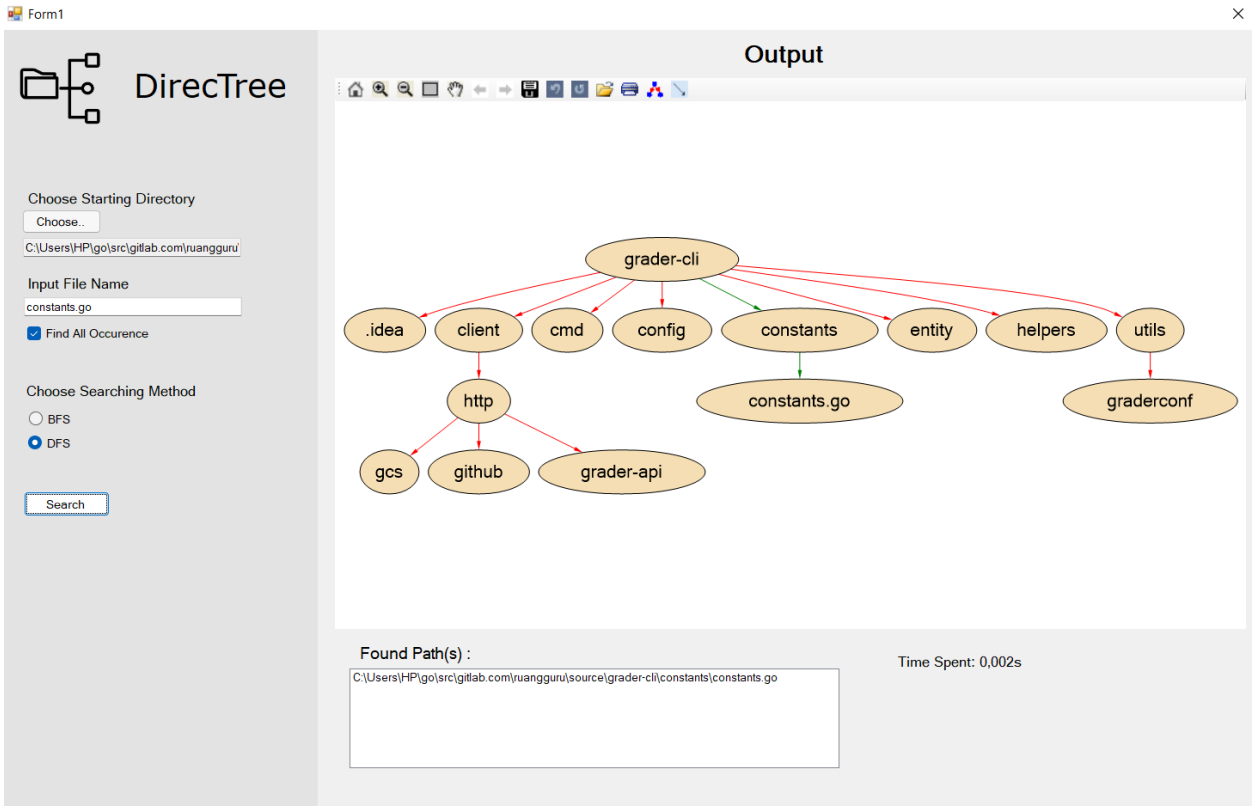


NB : algoritma kami melakukan pengecekan file terlebih dahulu sebelum melanjutkan searching, sehingga contoh garis merah pada bagian paling kiri merupakan hal yang mungkin terjadi.

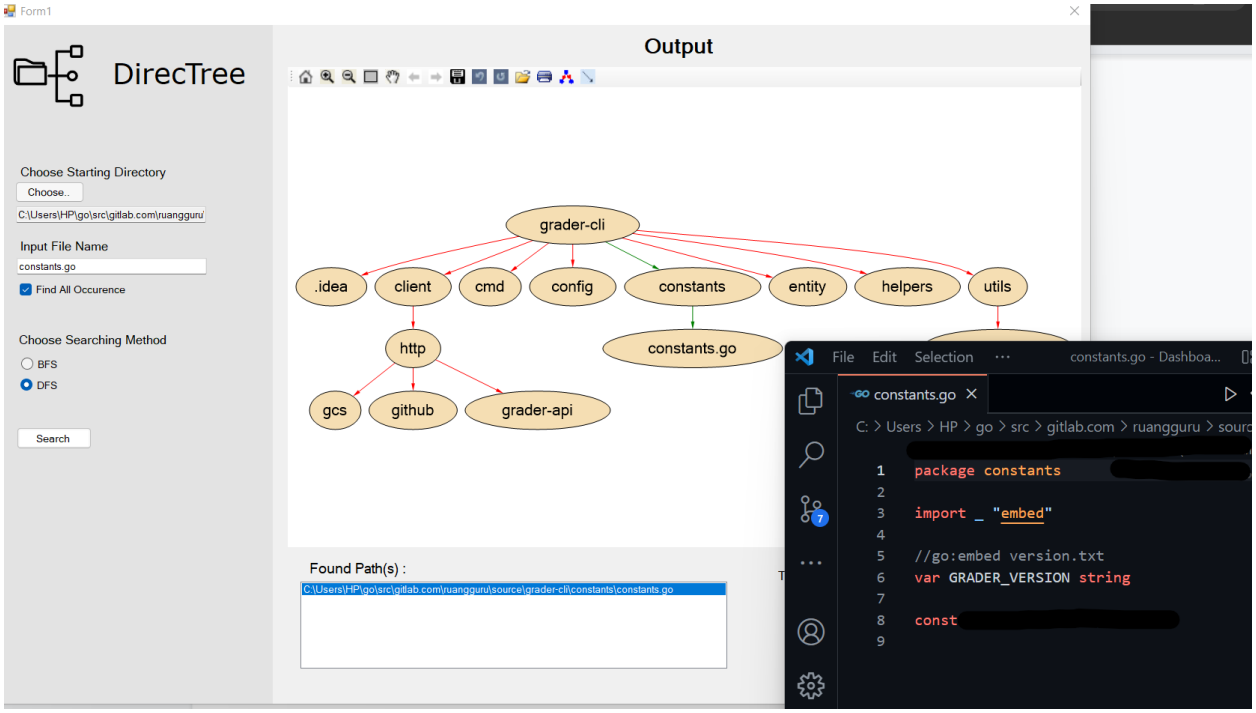
DFS



All Occurence DFS



Hyperlink test



5. Analisis desain solusi algoritma BFS dan DFS

   Dari kasus yang sudah dijalankan oleh kami sebelumnya, akan selalu ditemukan solusi dengan menggunakan algoritma DFS maupun BFS. Kedua algoritma tidak dapat dibandingkan karena mempunyai kompleksitas yang berbeda, dan terkadang hanya berbeda sepersekian detik. Akan tetapi, kita dapat melihat perbedaan waktu yang cukup signifikan untuk test case tertentu yang diberikan. Jika file yang dicari berada di titik terjauh (secara melebar) dari graph, maka akan lebih cepat menggunakan BFS. Namun jika file yang dicari berada jauh di dalam (secara vertikal) dalam graph, maka akan lebih cepat dengan algoritma DFS.

BAB 5

KESIMPULAN DAN SARAN

Dapat disimpulkan bahwa algoritma DFS dan BFS dapat digunakan untuk menyelesaikan implementasi dari aplikasi *folder crawling* ini. Meskipun algoritma ini bukan merupakan algoritma terbaik untuk menemukan file di folder tertentu, namun dapat dipastikan bahwa semua file dapat dicari, jika waktu yang dibutuhkan sudah memenuhi. Untuk tiap algoritma BFS dan DFS terdapat kelebihan dan kekurangannya masing-masing. Untuk algoritma DFS akan lebih efektif jika file yang dicari terletak lebih dalam pada graf yang sudah dibentuk, dan untuk algoritma BFS akan lebih efektif jika file yang dicari terletak lebih jauh melebar pada graf yang sudah dibentuk.

Penulis menyadari masih terdapat banyak kekurangan pada aplikasi *folder crawling* ini. Penulis berharap bahwa aplikasi ini dapat diimplementasi menggunakan algoritma yang lain yang lebih efektif daripada BFS dan DFS. Selain itu, dapat juga memperbagus tampilan dari GUI yang sudah disediakan agar lebih indah lagi.