Tugas Besar 2 IF2211 Strategi Algoritma

Semester II Tahun 2021/2022

Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder Crawling*

Dipersiapkan oleh:

Kelompok 6 (doraemonangis)

Nama	NIM
Taufan Fajarama Putrawansyah R	13520031
Raden Haryosatyo Wisjnunandono	13520070
Mohamad Daffa Argakoesoemah	13520118

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

DAFTAR ISI

BAB 1: DESKRIPSI TUGAS

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.

Dalam tugas besar ini, kami 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 kami inginkan. Kami juga diminta untuk memvisualisasikan hasil dari pencarian *folder* tersebut dalam bentuk pohon.

Selain pohon, kami 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*.

BAB 2: LANDASAN TEORI

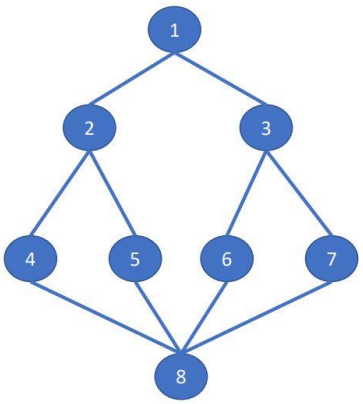
2.1 *Graf Traversal*

Graf traversal mengacu pada proses mengunjungi (memeriksa dan/atau memperbarui) setiap simpul dalam graf hingga membentuk suatu lintasan. Lintasan tersebut diklasifikasikan menurut urutan simpul yang dikunjungi. Jika setiap simpul dalam suatu graf akan dilintasi oleh algoritma berbasis pohon (seperti BFS atau DFS), maka algoritma tersebut harus dipanggil setidaknya sekali untuk setiap simpul yang masih belum dikunjungi saat diperiksa.

2.2 *Breadth First Search (BFS)*

Algoritma pencarian melebar (*Breadth First Search*/BFS) dilakukan secara traversal dimulai dari simpul *v* dan mengunjungi simpul-simpul tetangganya terlebih dahulu. Setelah semua simpul tetangga *v* sudah dikunjungi, pencarian dilanjutkan ke setiap simpul tetangga *v* dan mengunjungi simpul-simpul tetangganya (jika pohon traversal berarti simpul yang dimaksud adalah simpul anaknya). Pencarian tersebut dilakukan hingga tujuan pencarian tercapai atau semua simpul sudah dikunjungi. Ilustrasinya sebagai berikut.

BFS: Ilustrasi

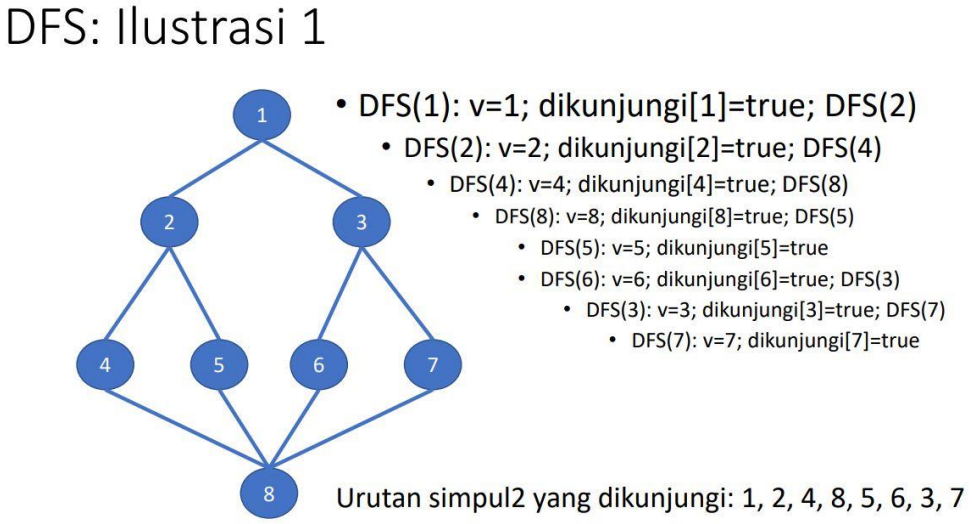


Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

Urutan simpul2 yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

2.3 *Depth First Search (DFS)*

Algoritma pencarian mendalam (*Depth First Search*/DFS) dilakukan secara traversal dimulai dari simpul *v* dan memilih satu simpul tetangga *v* untuk dikunjungi hingga simpul tersebut tidak memiliki simpul tetangga lagi untuk dikunjungi (dalam pohon traversal berarti simpul yang dimaksud adalah simpul anaknya). Setelah itu, pencarian dilanjutkan dengan runut-balik (*backtrack*) ke simpul sebelumnya yang mempunyai simpul tetangga yang belum dikunjungi. Pencarian tersebut dilakukan hingga tujuan pencarian tercapai atau semua simpul sudah dikunjungi. Ilustrasinya sebagai berikut.



2.4 C# Desktop Application Development

C# merupakan salah satu bahasa pemrograman berorientasi objek yang biasanya digunakan untuk pemrograman *server-side website*, membangun aplikasi *desktop* ataupun *mobile*, pemrograman dan sebagainya. Sama seperti pemrograman berorientasi objek lainnya, C# juga menerapkan konsep-konsep objek seperti *inheritance*, *class*, *polymorphism*, *encapsulation*, dan lain-lain. C# juga membutuhkan IDE dan *Framework*. Adapun IDE dan *Framework* yang digunakan adalah Visual Studio dengan *Framework* .NET.

Untuk pengembangan aplikasi *desktop*, terdapat berbagai jenis aplikasi yang disediakan oleh Visual Studio antara lain Console Application, Windows Forms Application, WPF Application, dan sebagainya. Setiap jenis aplikasi memiliki kekurangan dan kelebihannya masing-masing yang penggunaannya disesuaikan dengan kebutuhan pengguna. Pada tugas besar kali ini digunakan Windows Forms Application untuk merealisasikan GUI (*Graphical User Interface*) dari program yang dibuat.

BAB 3: ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Secara garis besar, berikut langkah-langkah pemecahan masalah dan tahapan algoritma untuk memberikan fitur yang diinginkan:

1. Program menerima *starting directory* dan *goal directory* yang ingin dicari. Program juga memeriksa apakah pencarian dilakukan secara menyeluruh atau hanya mencari satu *goal directory*.
2. Program menerima pilihan metode pencarian yang akan dilakukan (BFS/DFS) dan akan memetakan file dan folder menjadi elemen algoritma BFS/DFS.
3. Pencarian pada simpul tetangga di kedalaman yang sama diprioritaskan untuk memeriksa file terlebih dahulu kemudian memeriksa folder sesuai aturan pencarian yang dipilih (BFS/DFS).
4. Apabila pencarian dilakukan hanya untuk mencari satu *goal directory*, maka program akan berhenti ketika pertama kali berhasil menemukan *goal directory*. Jika pencarian dilakukan secara menyeluruh, maka program akan berhenti ketika semua simpul dari pohon traversal yang terbentuk sudah dikunjungi.
5. Setelah program selesai melakukan pencarian, program akan menampilkan lintasan graf berbentuk pohon traversal dimulai dari *starting directory* sebagai *root* hingga mencapai solusinya.

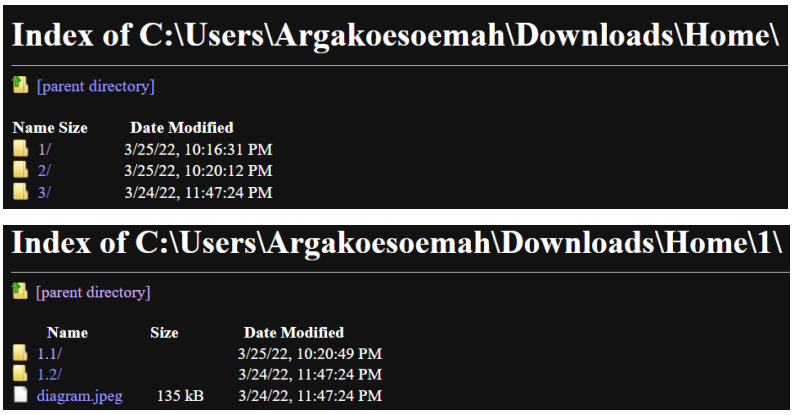
3.2 Pemetaan Elemen Algoritma BFS dan DFS

Pada permasalahan tugas besar ini, *starting directory* direpresentasikan sebagai simpul akar (*root*) dari pohon traversal. Setiap folder dan file di dalam *starting directory* direpresentasikan sebagai simpul tetangga (*adjacent vertex/node*) dan lintasan pencarian sebagai sisi (*edge*). Sehingga nantinya akan terbentuk pohon traversal yang merupakan hasil dari *Folder Crawling*.

3.3 Contoh Ilustrasi Kasus Lain

Berikut kasus apabila ada nama file/folder yang sama dengan simpul *parent*-nya. Selain itu, terdapat folder yang tidak ada file di dalamnya.

Data test:



Index of C:\Users\Argakoesoemah\Downloads\Home\1\1.1

[parent directory]

Name	Size	Date Modified
diagram.jpeg	135 kB	3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\1\1.2

[parent directory]

Name	Size	Date Modified
not.jpeg	135 kB	3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\1\1.2

[parent directory]

Name	Size	Date Modified
not.jpeg	135 kB	3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\2

[parent directory]

Name	Size	Date Modified
2.1/		3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\2\2.1

[parent directory]

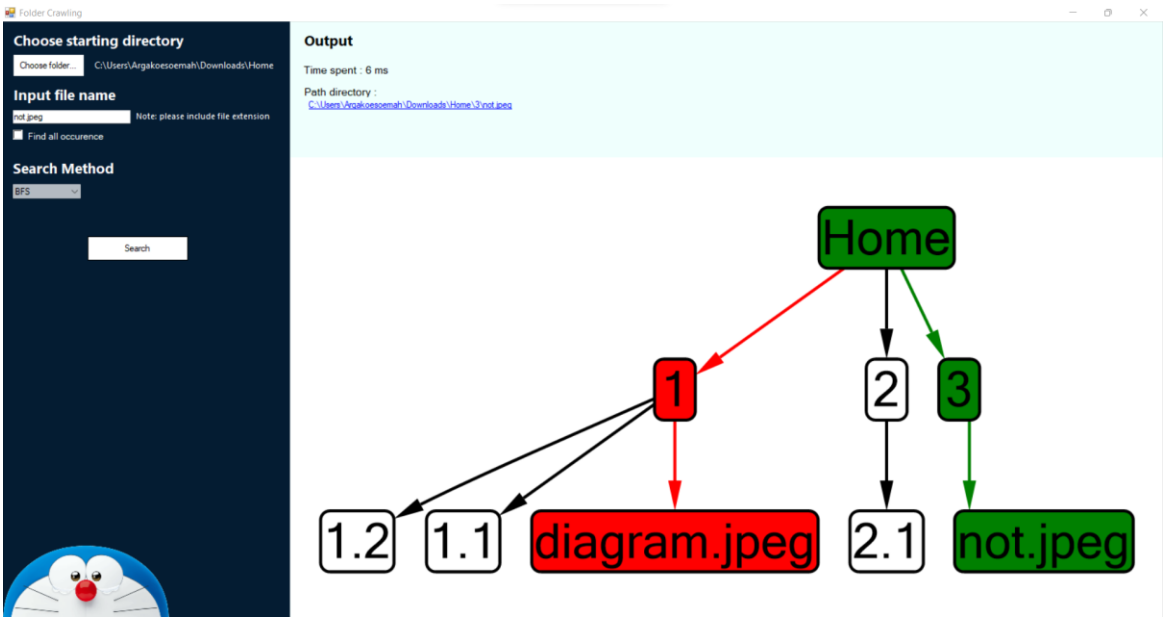
Name	Size	Date Modified
not.jpeg	135 kB	3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\3

[parent directory]

Name	Size	Date Modified
not.jpeg	135 kB	3/24/22, 11:47:24 PM

Hasil pencarian:



BAB 4: IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

**note: pada bagian ini hanya dilampirkan algoritma untuk searching saja. Untuk algoritma lengkap searching, pembentukan, dan pewarnaan graf dapat dilihat pada source code.*

Berikut implementasi algoritma BFS dalam *pseudocode*:

```
function BFS (startingDir, fileDicari : string, isCariSemua: boolean) ->
graph
KAMUS LOKAL
    antian : Queue
    dikunjungi : array of string
    found : boolean
    currentDirectory : string


ALGORITMA
    antrian.Enqueue(startingDir)
    dikunjungi.Add(startingDir)
    found <- false

    while (antrian tidak kosong) do
        currentDirectory = antrian.Dequeue()
        if (tidak ada isi dari currentDirectory) then
            warnaiMerah(currentDirectory)
        else
        // traversal tiap file yang ada di currentDirectory
            for tiap file yang ada di currentDirectory do


            //jika nama file sama dengan fileDicari
                if file = fileDicari then
                    found <- true
                    graph.TambahEdge(currentDirectory, file)
                    graph.warnaNode(file) <- hijau


                    if (not isCariSemua)then
                        break // hentikan pencarian

                else
                    graph.TambahEdge(currentDirectory, file)
                    graph.warnaNode(file) <- merah
            endfor

            // jika belum ditemukan file pada currentDirectory atau memilih
opsi cari semua file
            if (not found or isCariSemua) then

            //traversal seluruh directory yang ada di currentDirectory
```

```
            for (semua directory di currentDirectory) do
                graph.TambahEdge(currentDirectory, directory)
                antian.Enqueue(directory)
            endfor
        endwhile
```

Berikut implementasi algoritma DFS dalam *pseudocode*:

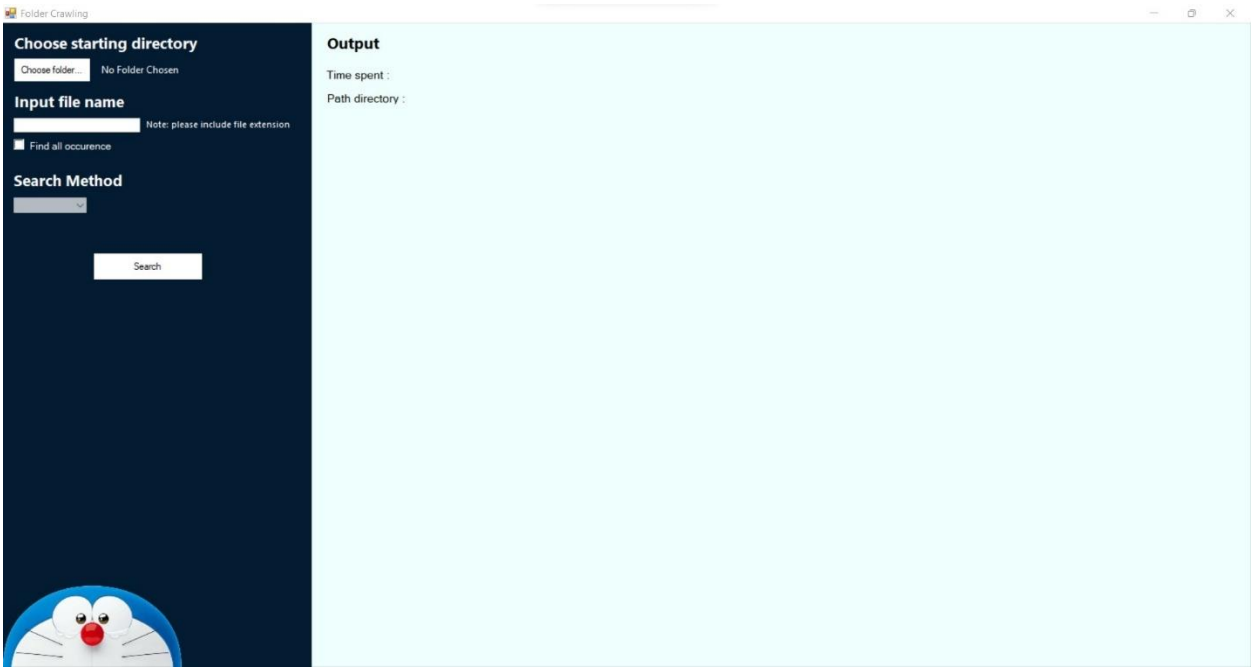
```
function DFS (startingDir, fileDicari : string, isCariSemua, found: boolean)
-> graph


KAMUS LOKAL
  found : boolean
  dikunjungi : array of boolean
ALGORITMA
  // Basis: file ditemukan (untuk not isAll Occurence)
  // atau pencarian berakhir untuk semua file dan directory


  if (tidak ada simpul tetangga dari startingDir) then
    warnaiMerah(startingDir)
     for (tiap file dalam startingDir) do
    if (not found) then


    // file merupakan file yang dicari ditemukan
      if (file = fileDicari) then
        warnaiHijau(file)
        graf.TambahEdge(startingDir, file)


        if (not isCariSemua) then
          found <-true


    // file bukan merupakan file yang dicari
      else
        warnaiMerah(file)
        graf.TambahEdge(startingDir, file)

    // jika belum menemukan file atau  dicari all Occurence maka
    if (not found or isCariSemua) then
      dikunjungi[startingDir] <- false
      for tiap directory dalam startingDir do
        if (dikunjungi[directory] == false)
          graph.TambahEdge(startingDir, directory)


          //rekurens
```

```
        DFS(directory)
    endfor
```

4.2 Struktur Data yang Digunakan

Berikut adalah diagram kelas pada program ini:

4.3 Tata Cara Penggunaan Program



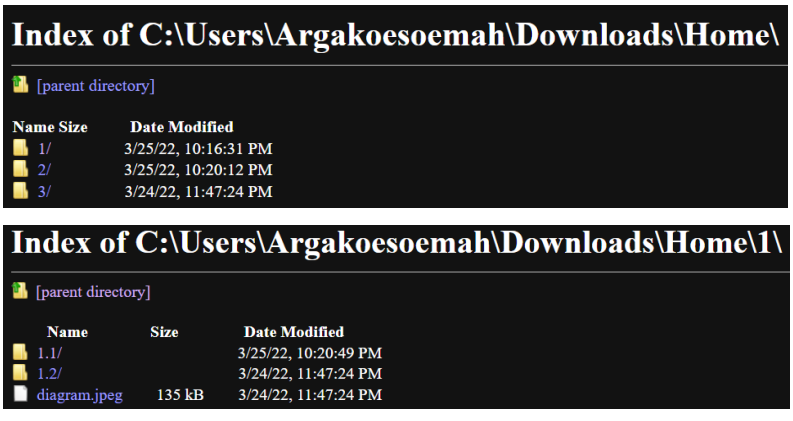
1. Pilih starting directory dengan cara mengklik tombol Choose folder.
2. Masukkan nama file yang ingin dicari pada textbox
3. Centang pilihan “Find all occurrence” jika ingin mencari semua kemungkinan yang memiliki nama file sama persis
4. Klik search untuk mulai pencarian

4.4 Hasil Pengujian

a. Pengujian pertama

Pencarian dengan metode BFS dan opsi satu file saja

Data test:



Index of C:\Users\Argakoesoemah\Downloads\Home\1\1.1

[parent directory]

Name	Size	Date Modified
diagram.jpeg	135 kB	3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\1\1.2

[parent directory]

Name	Size	Date Modified
not.jpeg	135 kB	3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\1\1.2

[parent directory]

Name	Size	Date Modified
not.jpeg	135 kB	3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\2

[parent directory]

Name	Size	Date Modified
2.1/		3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\2\2.1

[parent directory]

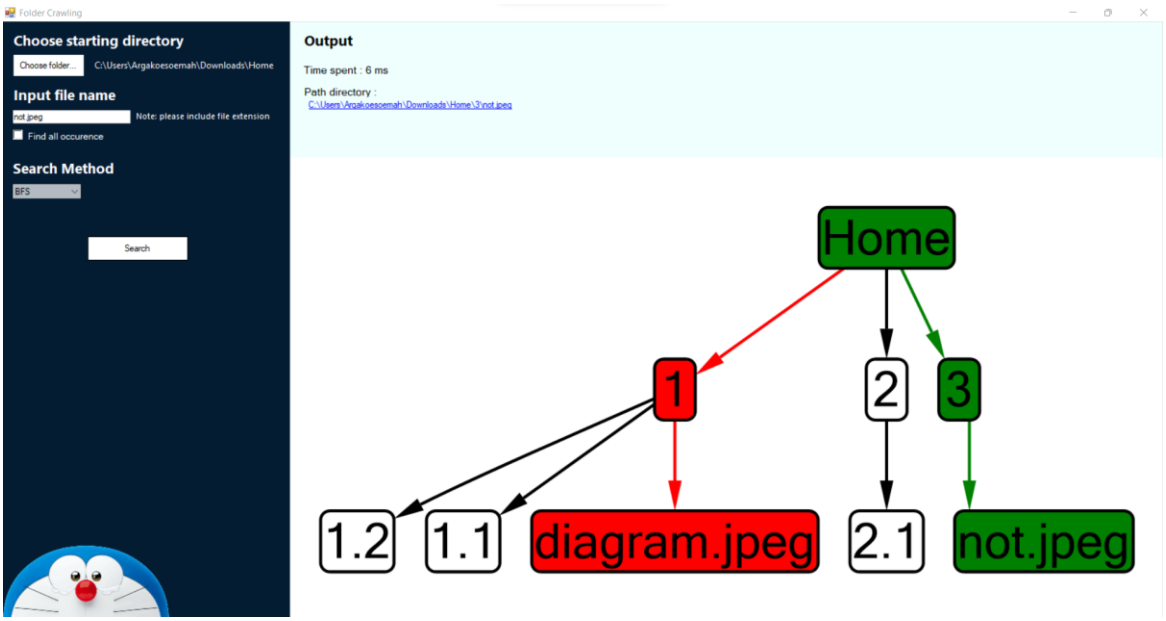
Name	Size	Date Modified
not.jpeg	135 kB	3/24/22, 11:47:24 PM

Index of C:\Users\Argakoesoemah\Downloads\Home\3

[parent directory]

Name	Size	Date Modified
not.jpeg	135 kB	3/24/22, 11:47:24 PM

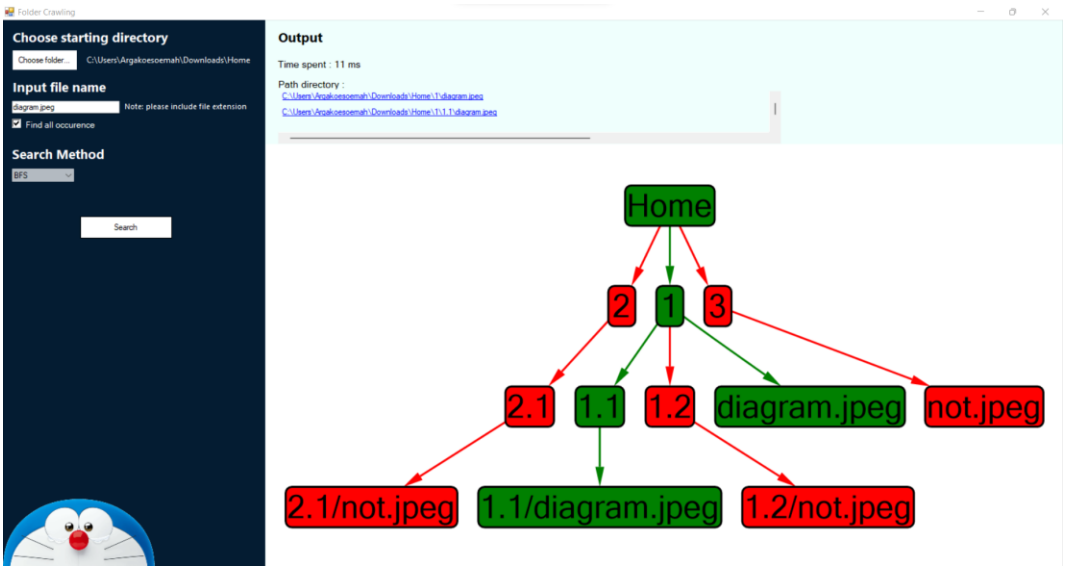
Hasil pencarian:



b. Pengujian kedua

Pencarian dengan metode BFS dan opsi semua nama file yang sama pada input query. Data test sama dengan pengujian pertama.

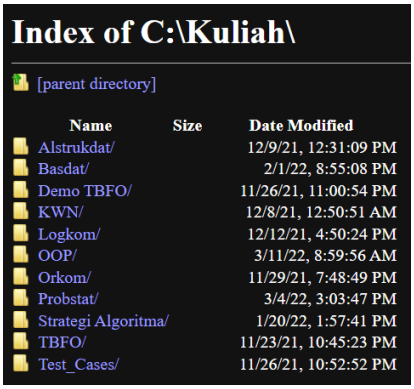
Hasil pencarian:

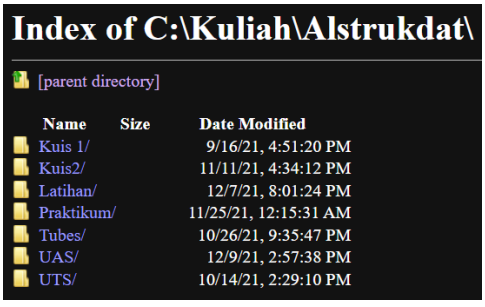


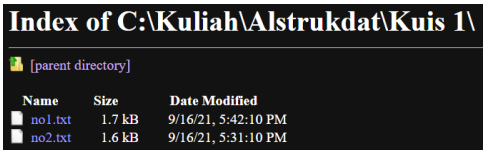
c. Pengujian ketiga

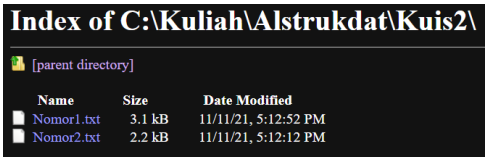
Pencarian dengan metode DFS dengan opsi satu file saja

Data test:

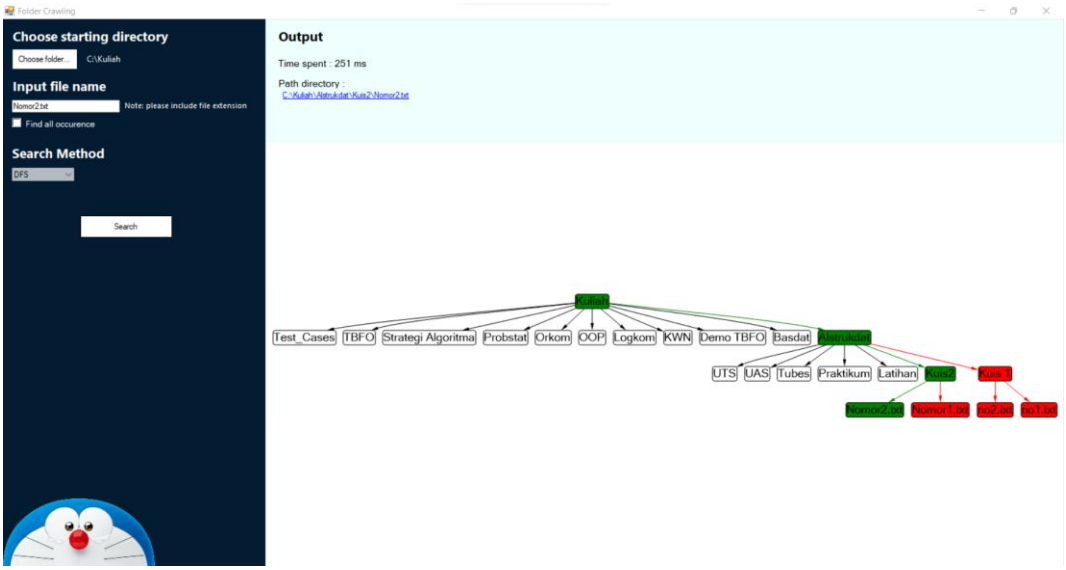






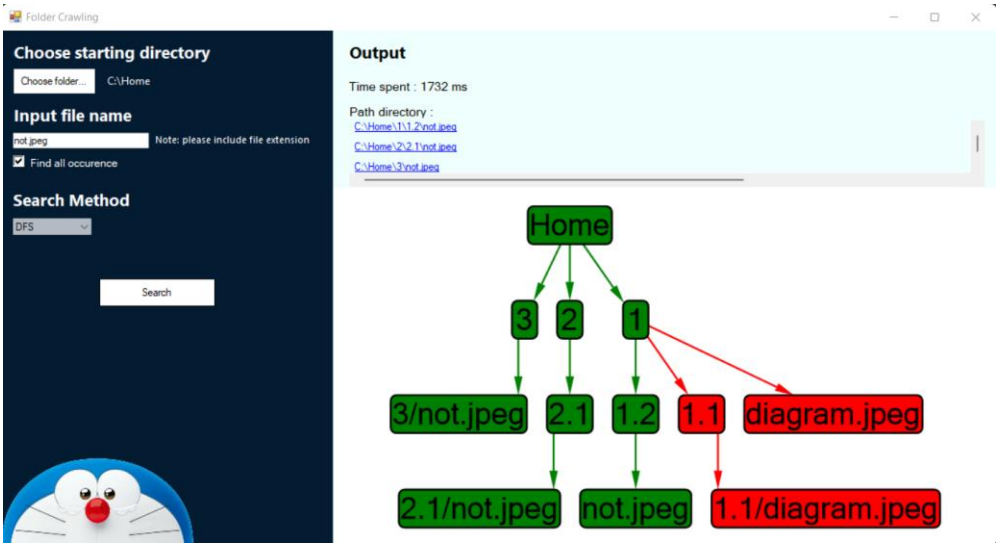


Hasil pengujian:



d. Pengujian keempat

Pencarian dengan metode DFS dan opsi semua nama file yang sama pada input query. Data test sama dengan pengujian pertama.



4.5 Analisis Desain Solusi

Pada pengujian pencarian dengan BFS dan DFS diatas kita dapat melihat bahwa kedua algoritma sama sama dapat mencari file yang diinginkan. Hanya saja terdapat perbedaan yang cukup fundamental dalam proses pencarian dimana BFS menggunakan while loop sementara DFS menggunakan rekursif.

Pada kasus pencarian seluruh elemen BFS lebih efektif karena nature dari algoritmanya yang mengiterasi seluruh elemen-elemen yang ada yang dimasukkan kedalam antrian. Sementara itu dalam kasus pencarian satu buah saja lebih efektif DFS karena dapat mencapa base case apabila telah menemukan folder yang dicari.

Lebih dari itu DFS lebih cepat jika mencari file yang letaknya lebih mendalam (level pohon semakin tinggi) sementara BFS lebih cepat jika file yang dicari letaknya tidak dalam (level pohon rendah).

BAB 5: KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah mengimplementasi algoritma DFS dan BFS dalam aplikasi folder crawler kami mendapat beberapa pelajaran sebagai berikut:

1. Algoritma DFS dan BFS dapat digunakan dalam mencari file dalam sebuah directory (foler crawler)
2. Algoritma DFS dan BFS dapat digunakan untuk memvisualiasi pencarian folder crawler
3. Aplikasi juga berhasil menampilkan pohon pencarian beserta hyperlink menuju folder tempat file tersebut terletak.

5.2 Saran

Terdapat beberapa pengembangan yang dapat dilakukan terhadap aplikasi kami. Terutama pada algoritma DFS dan BFS yang dapat dibuat lebih efisien dengan tidak menggunakan fungsi yang sama berulang-ulang dan penyederhanaan struktur data yang dipakai sehingga bisa dilakukan efisiensi kinerja. GUI program juga dapat dibuat dengan tampilan yang lebih menarik terutama pada output dari graf yang terbentuk.

DAFTAR PUSTAKA

Munir, R. & Maulidevi, N. U. (2021). *Breadth/Depth First Search (BFS/DFS) Bagian 1*. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf. Diakses pada tanggal 23 Maret 2022.

Munir, R. & Maulidevi, N. U. (2021). *Breadth/Depth First Search (BFS/DFS) Bagian 2*. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf. Diakses pada tanggal 23 Maret 2022.

LAMPIRAN

Link Github: https://github.com/roastland/Tubes2_13520031.git
Link Video Youtube: https://www.youtube.com/watch?v=7chss6tPds4