LAPORAN TUGAS BESAR II
IF2211 STRATEGI ALGORITMA

Laporan dibuat untuk memenuhi salah satu tugas mata kuliah

IF2211 Strategi Algoritma

Disusun oleh:

Kelompok CrawlingBackToYou

13520014 Muhamamd Helmi Hibatullah

13520026 Muhammad Fajar Ramadhan

13520045 Addin Nabilal Huda

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER 2 TAHUN 2021/2022

DAFTAR ISI

BAB I

DESKRIPSI TUGAS

Hampir seluruh sistem operasi sudah menyediakan fitur pencarian 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 mencari file yang sesuai dengan query yang dimasukkan pengguna mulai dari *starting directory* hingga seluruh *children* dari *starting directory* tersebut sampai satu file pertama/seluruh file ditemukan atau tidak ada file yang ditemukan.

Dalam tugas besar ini, dibangun 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), aplikasi dapat menelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang pengguna inginkan. Aplikasi juga dapat memvisualisasikan hasil dari pencarian folder tersebut dalam bentuk pohon. Selain pohon, aplikasi juga dapat 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 II

LANDASAN TEORI

Traversal Graf

Algoritma traversal graf yaitu algoritma pencarian solusi pada graf dimana pencarian solusi dilakukan dengan mengunjungi suatu simpul dengan cara yang sistematik. Algoritma pencarian solusi terbagi menjadi dua, yaitu pencarian tanpa informasi (*uninformed/blind search*) dan pencarian dengan informasi (*informed search*). Pada *blind search*, tidak ada informasi tambahan pada simpul tujuan selain yang disediakan di definisi permasalahan, contohnya adalah algoritma *breadth first search*, *depth first search, depth limited search,* dan lain-lain. Sedangkan pada *informed search* mempunyai informasi pada simpul tujuan yang dapat mempercepat pencarian, contohnya adalah algoritma *best first search* dan A*.

Dalam proses pencarian solusi, terdapat dua representasikan graf, yaitu graf statis, graf yang sudah terbentuk sebelum proses pencarian dilakukan dan direpresentasikan sebagai struktur data, dan graf dinamis, graf yang terbentuk saat proses pencarian dilakukan.

Algoritma *Breadth First Search* (BFS)

Algoritma *Breadth First Search* adalah algoritma pencarian tanpa informasi dimana pencarian dilakukan dengan mengunjungi simpul-simpul pada graf yang bertetanggaan dengan simpul yang dipilih terlebih dahulu. Misalkan pencarian dimulai dari simpul v. Berikut langkah-langkah pencarian menggunakan algoritma BFS.

1. Kunjungi simpul v.
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

Algoritma *Depth First Search* (DFS)

Algoritma *Depth First Search* adalah algoritma pencarian tanpa informasi dimana pencarian dilakukan dengan mengunjungi satu simpul yang bertetanggan dengan simpul yang dipilih, kemudian penelusuran dilanjutkan ke simpul yang bertetanggaan dengan simpul sebelumnya sampai tidak ada lagi simpul yang bertetangga yang belum dikunjungi. Misalkan pencarian dimulai dari simpul v. Berikut langkah-langkah pencarian menggunakan algoritma DFS.

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 sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Pengembangan Aplikasi Desktop Menggunakan C#

Bahasa C# adalah sebuah bahasa pemrograman berorientasi objek yang disediakan oleh Microsoft dan berjalan pada *framework* .NET. .NET sendiri merupakan platform pengembang *open source*, yang dibuat oleh Microsoft, untuk membangun berbagai jenis aplikasi. Para pengembang dapat menulis aplikasi .NET dalam bahasa C#, F#, Visual C++, atau Visual Basic. Dengan adanya C#, para pengembang dapat mengembangkan berbagai jenis aplikasi yang kokoh dan aman, seperti aplikasi Windows, aplikasi web, aplikasi terdistribusi, aplikasi layanan web, aplikasi basis data, dan lain-lain. Bahasa pemrograman C# dibuat berdasarkan C++ dan Java, tetapi sudah ditambah banyak ekstensi untuk melakukan pendekatan pemrograman berorientasi komponen.

BAB III

ANALISIS PEMECAHAN MASALAH

Langkah-langkah Pemecahan Masalah

Pemecahan masalah pada tugas besar kali ini kami awali dengan menganalisis permasalahan dan mencoba untuk memetakan objek pencarian (yang kali ini adalah file dan folder) ke dalam bentuk pohon. Untuk dapat menemukan file yang dicari, program yang kami buat menerapkan algoritma pencarian tanpa informasi DFS dan BFS. Dalam proses pencarian solusi, program menggunakan pendekatan graf dinamis yaitu graf (atau dalam kasus ini direpresentasikan sebagai pohon) yang terbentuk saat proses pencarian dilakukan. Berikut langkah-langkah dalam melakukan pencarian yang terbagi menjadi beberapa bagian.

a. Pencarian dengan Algoritma BFS (*Breadth First Search*)
    1. Inisialisasi folder root sebagai akar, tambahkan folder tersebut ke antrian folder yang belum dikunjungi paling belakang
    2. Bangkitkan folder pada antrian folder yang belum dikunjungi terdepan
    3. Periksa apakah pada folder tersebut terdapat file dengan nama file yang dicari.
    4. Jika ditemukan nama file yang sesuai, simpan alamat lengkap file tersebut sebagai solusi dan kembalikan solusi. Jika program diminta untuk menemukan semua kemunculan maka lanjutkan pencarian.
    5. Jika tidak ditemukan atau folder tidak mengandung file, tambahkan seluruh folder anak dari folder yang sedang dibangkitkan ke antrian folder yang belum dikunjungi paling belakang
    6. Ulangi langkah 2-4 hingga seluruh folder pada antrian sudah dikunjungi atau file sudah ditemukan pada kasus pencarian 1 file (bukan all occurence)
b. Pencarian dengan Algoritma DFS (*Depth First Search*)
    1. Inisialisasi folder root sebagai akar, tambahkan folder tersebut ke stack (tumpukan) folder yang belum dikunjungi
    2. Bangkitkan folder paling atas (pop) pada stack folder yang belum dikunjungi
    3. Periksa apakah pada folder tersebut menyimpan file dengan nama file yang dicari.

4. Jika ditemukan nama file yang sesuai, simpan alamat lengkap file tersebut sebagai solusi dan kembalikan solusi. Jika program diminta untuk menemukan semua kemunculan maka lanjutkan pencarian.
5. Jika tidak ditemukan file dalam folder tersebut atau folder tidak mengandung file, tambahkan (push) seluruh folder anak (adjacency folder) dari folder yang sedang dibangkitkan ke dalam stack folder yang belum dikunjungi.
6. Ulangi langkah 2 hingga 5 sampai tidak ada lagi folder yang berada dalam stack folder yang belum dikunjungi

c. Visualisasi Graf pada Pencarian dengan Algoritma BFS (*Breadth First Search*)
    1. Inisialisasi folder root sebagai akar, tambahkan folder tersebut ke antrian folder yang belum dikunjungi paling belakang
    2. Bangkitkan folder pada antrian folder yang belum dikunjungi terdepan
    3. Tambahkan edge antara node folder yang sedang dibangkitkan dan semua file yang berada di dalamnya. Warna hijau untuk file yang benar, merah untuk file yang salah
    4. Tambahkan edge antara node folder yang sedang dibangkitkan dan semua anak folder yang berada pada folder yang sedang dibangkitkan. Warna hijau untuk folder yang benar, merah untuk folder yang salah, dan hitam untuk folder yang belum dikunjungi/diperiksa
    5. Tambahkan seluruh folder anak dari folder yang sedang dibangkitkan ke antrian folder yang belum dikunjungi paling belakang jika pada algoritma pencarian folder sebelumnya folder tersebut sudah dikunjungi
    6. Ulangi langkah 2 hingga 5 sampai sudah tidak ada folder pada antrian
d. Visualisasi Graf Pencarian dengan Algoritma DFS (*Depth First Search*)
    1. Inisialisasi folder root sebagai akar, tambahkan folder tersebut ke stack (tumpukan) folder yang belum dikunjungi
    2. Bangkitkan folder paling atas (pop) pada stack folder yang belum dikunjungi
    3. Tambahkan edge antara node folder yang sedang dibangkitkan dan semua file yang berada di dalamnya. Warna hijau untuk file yang benar, merah untuk file yang salah

4. Tambahkan edge antara node folder yang sedang dibangkitkan dan semua anak folder yang berada pada folder yang sedang dibangkitkan. Warna hijau untuk folder yang benar, merah untuk folder yang salah, dan hitam untuk folder yang belum dikunjungi/diperiksa
5. Tambahkan seluruh folder anak dari folder yang sedang dibangkitkan ke stack (tumpukan) folder yang belum dikunjungi jika pada algoritma pencarian folder sebelumnya folder tersebut sudah dikunjungi

Pemetaan Elemen-elemen Algoritma BFS dan DFS

Dalam teknik pencarian file pada *Folder Crawling*, program kami menggunakan algoritma pencarian solusi BFS dan DFS dengan pendekatan graf dinamis. Karena struktur folder dan file merupakan sebuah hierarki, graf direpresentasikan sebagai pohon dengan folder root masukan user sebagai akar dari pohon. Setiap folder dan file direpresentasikan sebagai simpul dan hubungan bapak-anak pada folder direpresentasikan sebagai *edge*. Seluruh folder dan file yang dinaungi folder root tersebut membentuk ruang status. Pada proses pencarian, folder dengan file-file yang berada di dalamnya dilihat sebagai simpul-simpul pada level yang sama. Artinya, ketika suatu folder dikunjungi, file-file yang terdapat dalam folder tersebut sekaligus diperiksa. Namun, pada visualisasi graf, letak file-file tersebut adalah di level di bawah folder yang menaunginya (namun tetap dilihat sebagai level yang sama).

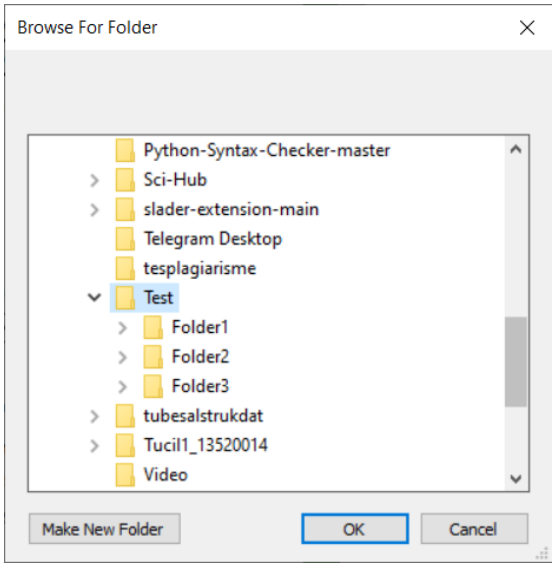
Daun pada pohon menyatakan solusi dari persoalan, yakni simpul tempat file ditemukan. Pada kasus pencarian seluruh kemunculan file, pohon dapat terdiri atas nol (file tidak ditemukan), satu, atau lebih dari satu daun. Sedangkan, pada kasus pencarian satu kemunculan file, pohon hanya dapat terdiri atas 0 (file tidak ditemukan) atau 1 daun. Pada program, daun-daun disimpan dalam struktur List of string bernama rightPath yang setiap daunnya direpresentasikan sebagai string path menuju file dari folder root. Daun-daun tersebut membentuk himpunan ruang solusi dari persoalan.

Pada proses pembangkitan status, struktur data berisi jalur dari simpul akar sampai ke simpul daun diperbaharui seiring dengan ditemukannya file. Setiap file ditemukan, program akan

menjalankan prosedur generateGraph() untuk dapat memperbaharui daftar *path* menuju file yang benar yang berisi simpul-simpul folder menuju file yang dicari.

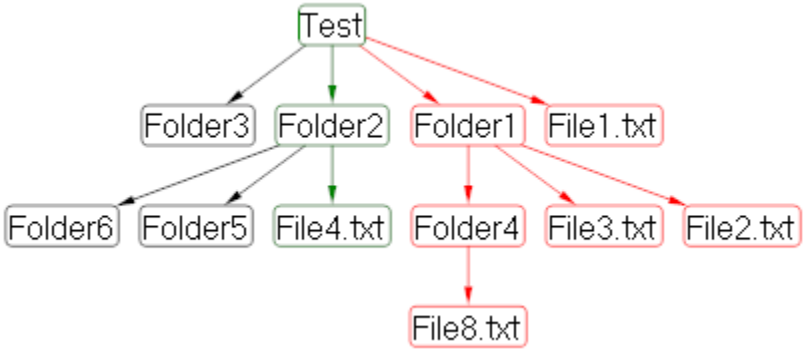
Contoh Ilustrasi Kasus Lain

Berikut adalah contoh pencarian yang kami lakukan dengan menggunakan program yang kami buat. Dibawah ini kami mencoba untuk mencari file bernama “File4.txt” di dalam folder Test.



Gambar 1 Contoh Input *Starting Directory*

Berikut adalah pohon dari hasil pencarian menggunakan algoritma *Depth First Search* (DFS).



Gambar 2 Pohon Hasil Pencarian dengan Algoritma DFS

Langkah *folder crawling* pada algoritma ini adalah sebagai berikut. Test → File1.txt → Test → Folder1 → File2.txt → Folder1 → File3.txt → Folder1 → Folder4 → File8.txt → Folder4 →

Folder1 → Test → Folder2 → File4.txt. Pada gambar, rute yang dilewati pada pencarian DFS diwarnai dengan warna merah dan rute untuk menuju tempat file berada diberi warna hijau. Rute yang sudah memasuki antrian tetapi belum diperiksa diberi warna hitam.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

Implementasi Program

Berikut beberapa atribut penting dari kelas Searching yang akan banyak digunakan pada implementasi prosedur penting di bagian selanjutnya.

- filePath: list of string. Untuk menampung satu atau lebih *path* dari file yang sudah ditemukan.
- rightPath: list of string. Untuk menampung satu atau lebih *path* dari folder root menuju file yang dicari.
- visitedPath: list of string. Untuk menampung satu atau lebih *path* dari semua *path* yang telah dikunjungi.

Berikut implementasi beberapa prosedur penting pada kelas Searching dalam bentuk *pseudocode*:

```
procedure DFS()
KAMUS
       folderStack: stack of Folder
       rootFolder: Folder
       currentFolder: Folder
       found: boolean

ALGORITMA
       tempStack.Push(folderRoot)
       while (folderStack tidak kosong dan (!found or cari seluruh kemungkinan)) do
              currentFolder <- folderQueue.Pop()
              visitedPath.Add(path directory dari currentFolder)

              if (currentFolder mengandung file yang dicari) then
                     filePath.Add(path file yang dicari pada current)
                     generatePath(currentFolder)
                     found <- true
```

```
            endif

            push semua folder anak dari currentFolder ke folderQueue
    endwhile
```

```
procedure BFS()
KAMUS
      folderQueue: queue of Folders
      rootFolder: Folder
      currentFolder: Folder
      found: boolean

ALGORITMA
      folderQueue.Enqueue(rootFolder)
      while (folderQueue tidak kosong dan (!found or cari seluruh kemungkinan)) do
            currentFolder <- folderQueue.Dequeue()
            visitedPath.Add(path directory dari currentFolder)

            if (currentFolder mengandung file yang dicari) then
                  filePath.Add(path file yang dicari pada current)
                  generatePath(currentFolder)
                  found <- true
            endif

            enqueue semua folder anak dari currentFolder ke folderQueue
      endwhile
```

```
procedure generatePath(input rightDir: Folder)
KAMUS
      currentDir: Folder

ALGORITMA
      currentDir <- rightDir
      while (currentDir != null)
```

```
            rightPath.Add(nama path dari currentDir)
            currentFolder <- parent dari currentFolder
        endwhile
```

Struktur Data

Berikut kelas-kelas yang dibuat beserta deskripsi, penjelasan atribut, dan *method*-nya.

a. Kelas Folder

Kelas Folder digunakan untuk menyimpan struktur data folder pada program. Kelas ini memiliki atribut-atribut sebagai berikut:

- listFile: atribut berupa array of string yang digunakan untuk menyimpan path dari file-file yang berada pada objek Folder.
- listFolder: atribut berupa array of string yang digunakan untuk menyimpan path dari folder-folder yang berada pada objek Folder.
- dirName: atribut berupa string yang digunakan untuk menyimpan path directory dari objek Folder.
- parent: atribut berupa Folder yang digunakan untuk menyimpan parent dari objek Folder.
- adjFolder: atribut berupa list of Folder yang digunakan untuk menyimpan folder-folder yang adjacent dengan objek Folder.
- visited: atribut boolean untuk menandai apakah objek Folder sudah dikunjungi/belum.

Tabel 1 Deskripsi Method Kelas Folder

Method	Deskripsi
Folder(string root)	Konstruktor Folder
void setVisited(Boolean val)	Mengganti atribut visited menjadi val
string[] getAllDir()	Mengembalikan atribut listFolder
string[] getAllFiles()	Mengembalikan atribut listFile
List<Folder> getAdj()	Mengembalikan atribut adjFolder
Boolean getVisited()	Mengembalikan atribut visited

string getDirname()	Mengembalikan atribut dirName
Folder getParent()	Mengembalikan atribut parent
Boolean checkFile(string name)	Mengembalikan true jika folder mengandung folder dengan nama name
string getFilePath(string filename)	Mengembalikan *path* lengkap dari file filename

b. Kelas Searching

Kelas Searching digunakan untuk melakukan operasi algoritma pencarian dengan DFS maupun BFS. Class ini memiliki atribut-atribut sebagai berikut:

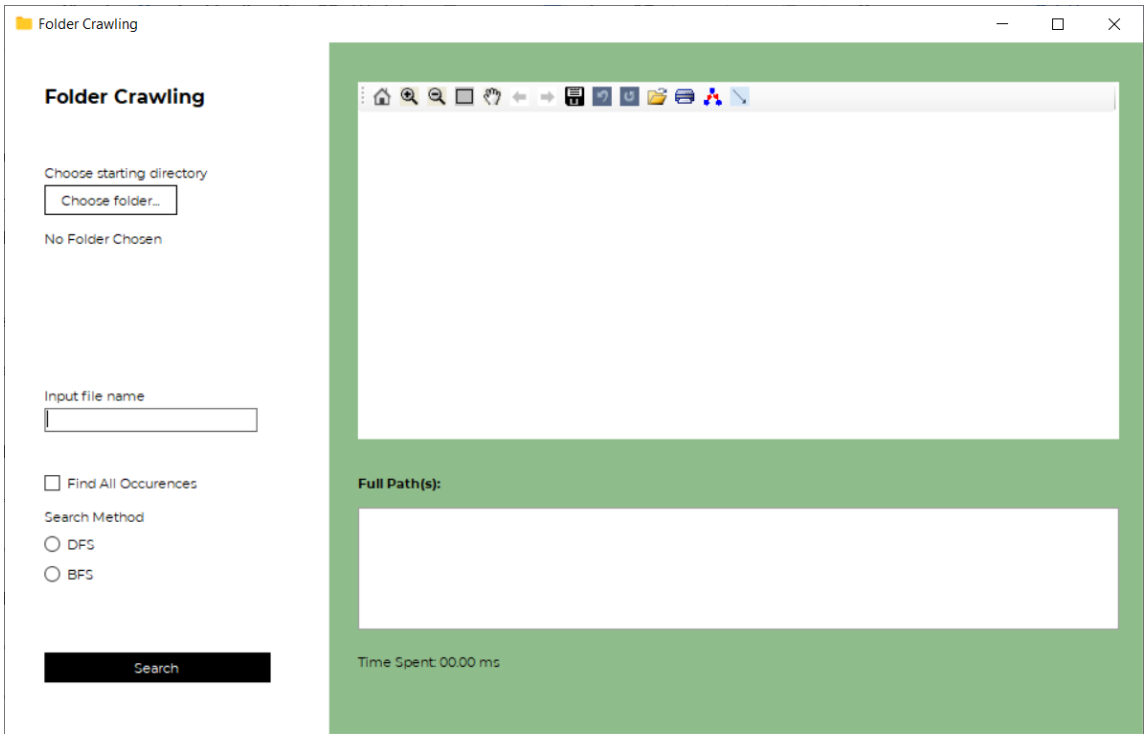
- isAllOccurence: atribut berupa boolean yang bernilai true bila user meminta program untuk mencari seluruh kemunculan file dalam folder.
- fileToSearch: atribut berupa string untuk menyimpan nama file yang dicari.
- rootPath: atribut berupa string untuk menyimpan *path* root.
- filePath: atribut berupa list of string untuk menampung satu atau lebih *path* dari file yang sudah ditemukan.
- rightPath: atribut berupa list of string untuk menampung satu atau lebih *path* dari folder root menuju file yang dicari.
- visitedPath: atribut berupa list of string untuk menampung satu atau lebih *path* dari semua *path* yang telah dikunjungi.

Tabel 2 Deskripsi Method Kelas Searching

Method	**Deskripsi**
Searching(string rootPath, string fileToSearch, bool isAllOccurence)	Konstruktor Searching
void DFS()	Melakukan pencarian dengan algoritma *Depth First Search* (DFS)
void BFS()	Melakukan pencarian dengan algoritma *Breadth First Search* (BFS)
List<String> getFilePath()	Mengembalikan atribut filePath
List<Folder> getRightPath()	Mengembalikan atribut rightPath
void generatePath(Folder rightDir)	Membuat path dari file ke root

Tata Cara Penggunaan Program

Untuk menggunakan program, pengguna cukup menjalankan program bernama "CrawlingBack2U.exe". Berikut tampilan awal pada saat program baru dijalankan.



Gambar 3 Tampilan Awal Program

Dapat dilihat dari gambar bahwa program ini memiliki fitur untuk memilih *starting directory*-nya sendiri, memasukkan nama file yang akan dicari, memilih untuk melakukan pencarian semua kemunculan, dan memilih metode pencarian. Pengguna juga dapat melihat graf yang terbentuk setelah pencarian dilakukan. Selain itu, disediakan juga daftar *path* lengkap dari file yang dicari. Jika pengguna mengklik *path* lengkap tersebut, akan terbuka File Explorer yang mengarah ke *path* tersebut. Namun, jika file yang dicari tidak ditemukan, kotak yang seharusnya menampilkan daftar *path* lengkap akan menampilkan pesan "File not found".

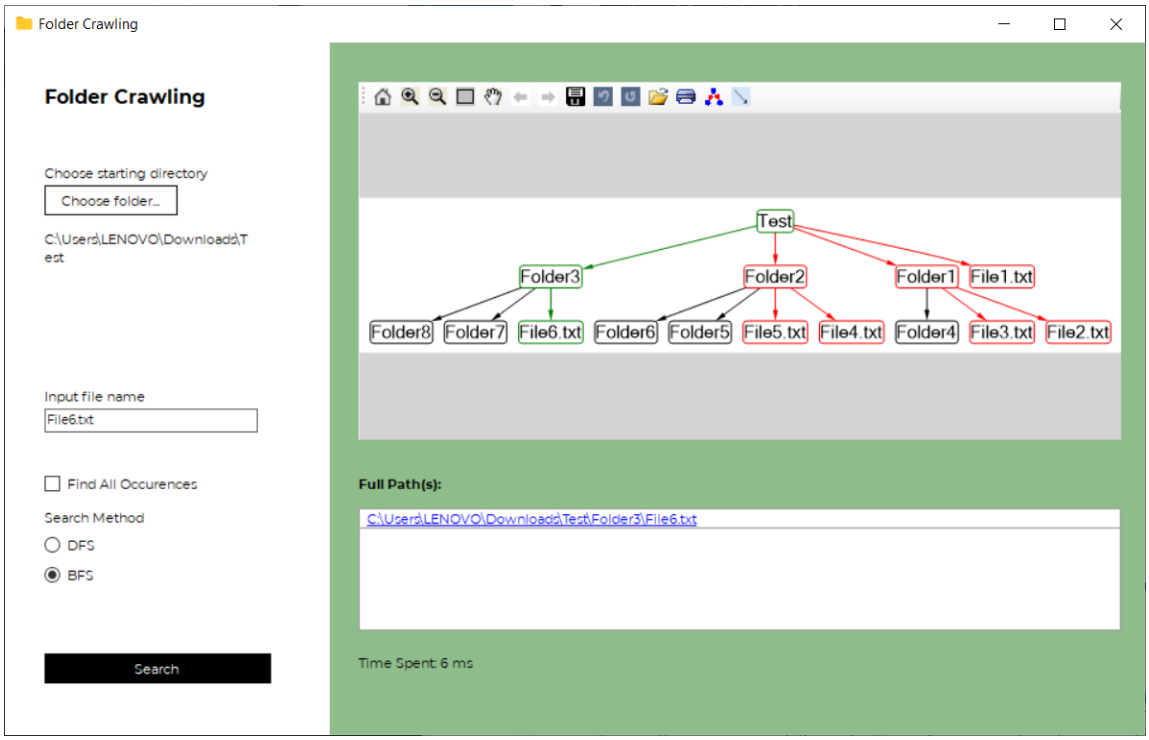
Untuk melakukan pencarian, pengguna diwajibkan untuk memilih *starting directory*, mengisi nama file yang akan dicari, dan memilih metode pencarian sebelum menekan tombol Search. Jika salah satu dari ketiga hal tersebut tidak dilakukan, tidak akan terjadi apapun ketika pengguna menekan tombol Search. Setelah ketiga hal tersebut terisi, pengguna boleh memilih

untuk mengklik *checkbox* Find All Occurence atau tidak. Jika *checkbox* ini diisi, pencarian yang akan dilakukan nanti adalah pencarian untuk semua kemunculan sehingga pohon folder yang terbentuk merupakan pohon folder yang lengkap. Sebaliknya, jika tidak diisi, pencarian akan berhenti ketika sudah menemukan file yang dicari pertama kali. Terakhir, pengguna dapat mengklik tombol Search untuk memulai pencarian. Setelah pencarian selesai, akan muncul pohon folder di kotak putih sebelah kanan dan *hyperlink* di bawahnya yang menunjukkan *path* lengkap dari hasil pencarian file. Pengguna dapat melakukan berbagai hal pada pohon dengan menggunakan *tool* yang dapat dipilih pada *toolbar* di dalam kotak. Pengguna juga dapat mengklik *hyperlink path* lengkap dari hasil pencarian untuk membuka folder tersebut dengan File Explorer.

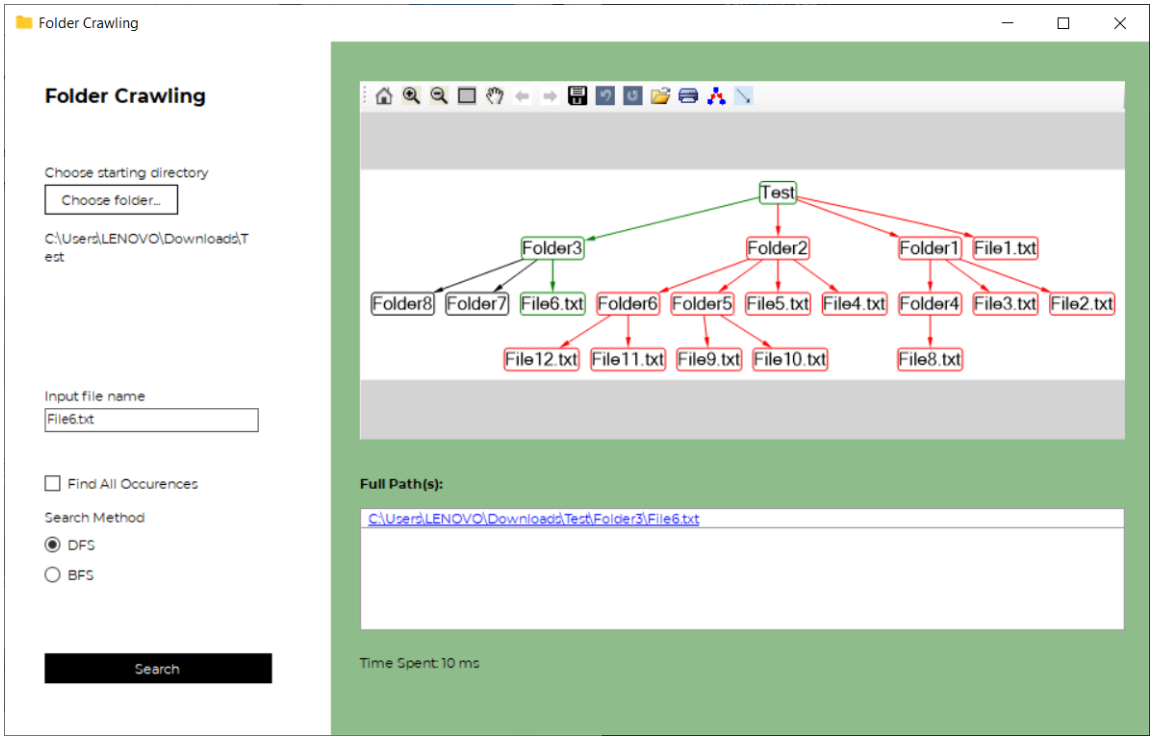
Analisis pada Pengujian

Pengujian dilakukan dengan mencoba berbagai kombinasi pohon folder. Dengan mengatur penempatan folder beserta file dan menekan tombol Search, proses pencarian pun berjalan.

a. Pengujian I



Gambar 4 Hasil Pengujian I dengan Algoritma BFS



Gambar 5 Hasil Pengujian I dengan Algoritma BFS

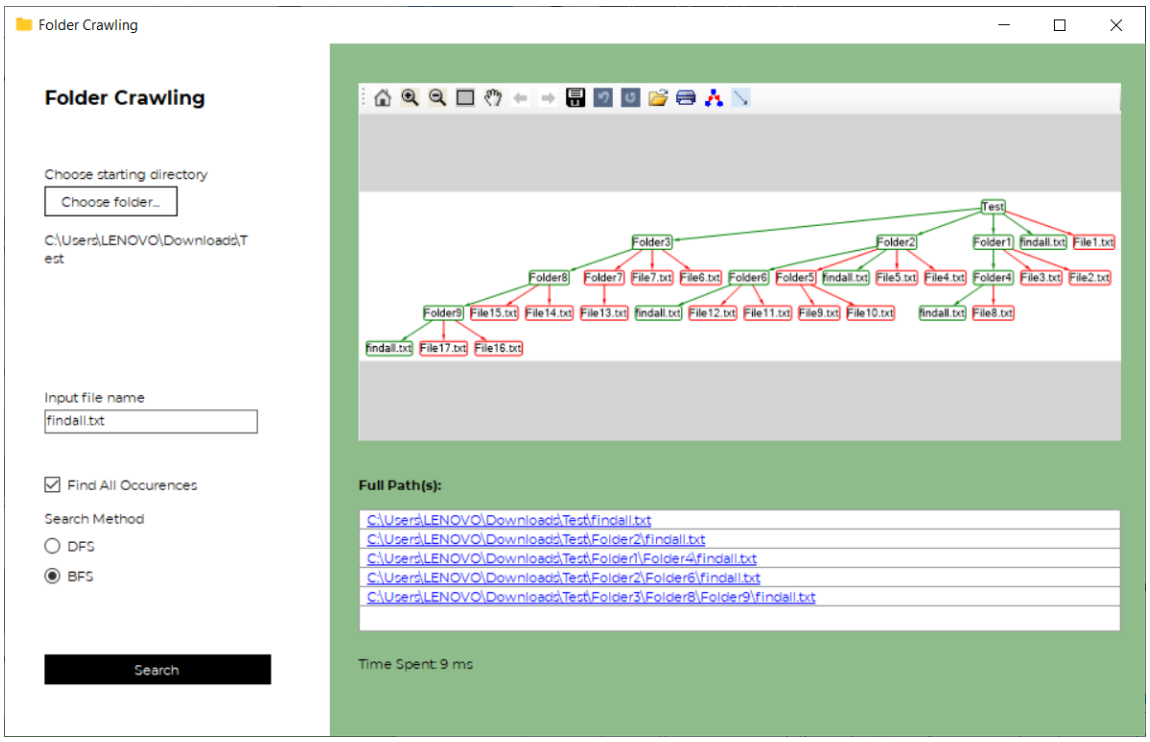
Pada pengujian I, dilakukan pencarian File6.txt pada folder Test dengan algoritma BFS dan DFS. Dapat terlihat bahwa pohon folder yang terbentuk dari hasil pencarian memiliki bentuk yang berbeda bergantung pada algoritma yang digunakan. Hal ini disebabkan karena algoritma DFS akan mencari file dengan menelusuri pohon folder hingga ke dasar terlebih dahulu, sedangkan pada algoritma BFS akan mencari file dengan menelusuri pohon per tingkat kedalaman.

Perhatikan pada Gambar 4, pada simpul Folder1, simpul File2.txt dan simpul File3.txt berwarna merah, sedangkan simpul Folder4 berwarna hitam. Perbedaan warna ini membuat seolah-olah pencarian pada Folder4 terlewat dan algoritma BFS yang dibuat mengalami sebuah kesalahan. Pada algoritma yang kami buat, saat mengunjungi suatu simpul folder, program akan mengecek file-file yang ada di folder tersebut terlebih dahulu. File-file ini tidak kita anggap sebagai keturunannya, melainkan sebagai bagian dari folder tersebut. Oleh karena itu, sebenarnya File2.txt dan File3.txt memiliki *level* yang sama dengan simpul Folder1, begitu juga dengan File4.txt dan File5.txt terhadap simpul Folder2. Setelah file-file pada sebuah folder dicek, barulah pencarian dilanjutkan

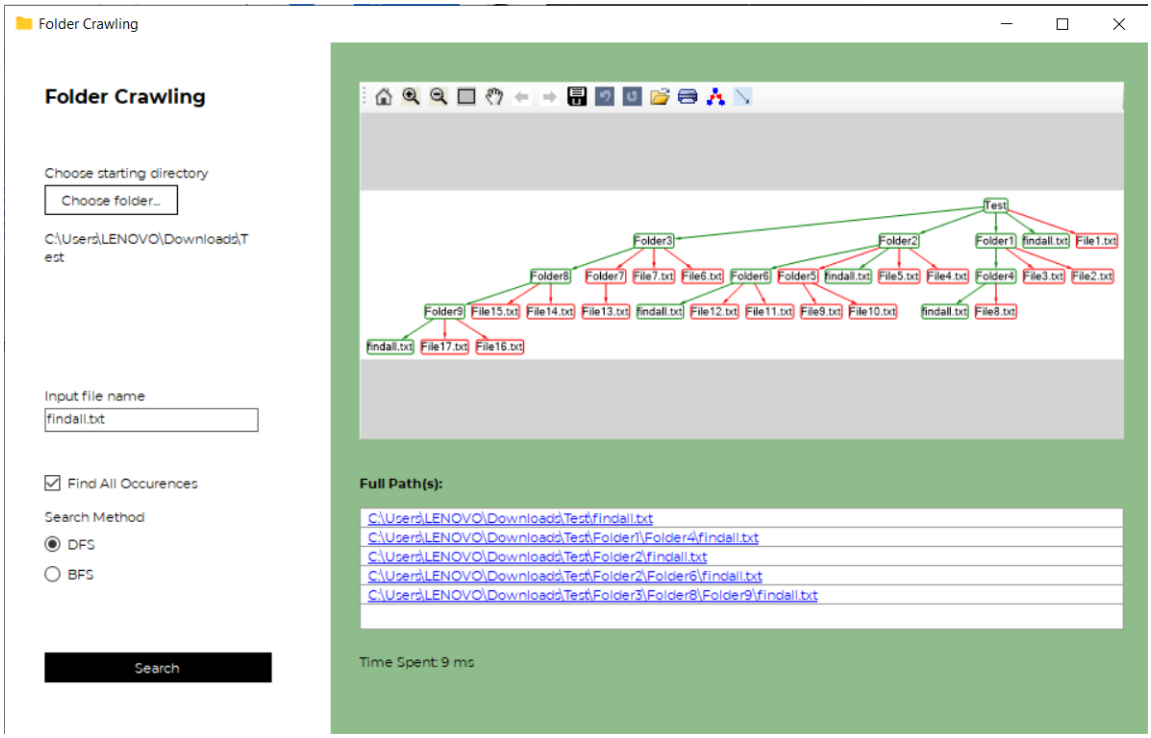
ke folder selanjutnya, entah itu ke folder anaknya atau ke folder lain, bergantung pada algoritma yang digunakan.

Dapat dilihat pada kedua gambar bahwa pencarian menggunakan algoritma BFS, pada kasus ini, memiliki waktu pencarian yang lebih cepat empat milisekon. Hal ini disebabkan karena pada algoritma BFS, jumlah perbandingan nama file yang dicari dengan nama file yang ada hanya sedikit, yaitu enam kali perbandingan. Sedangkan pada algoritma DFS, perbandingan dilakukan sebanyak sebelas perbandingan.

b. Pengujian II



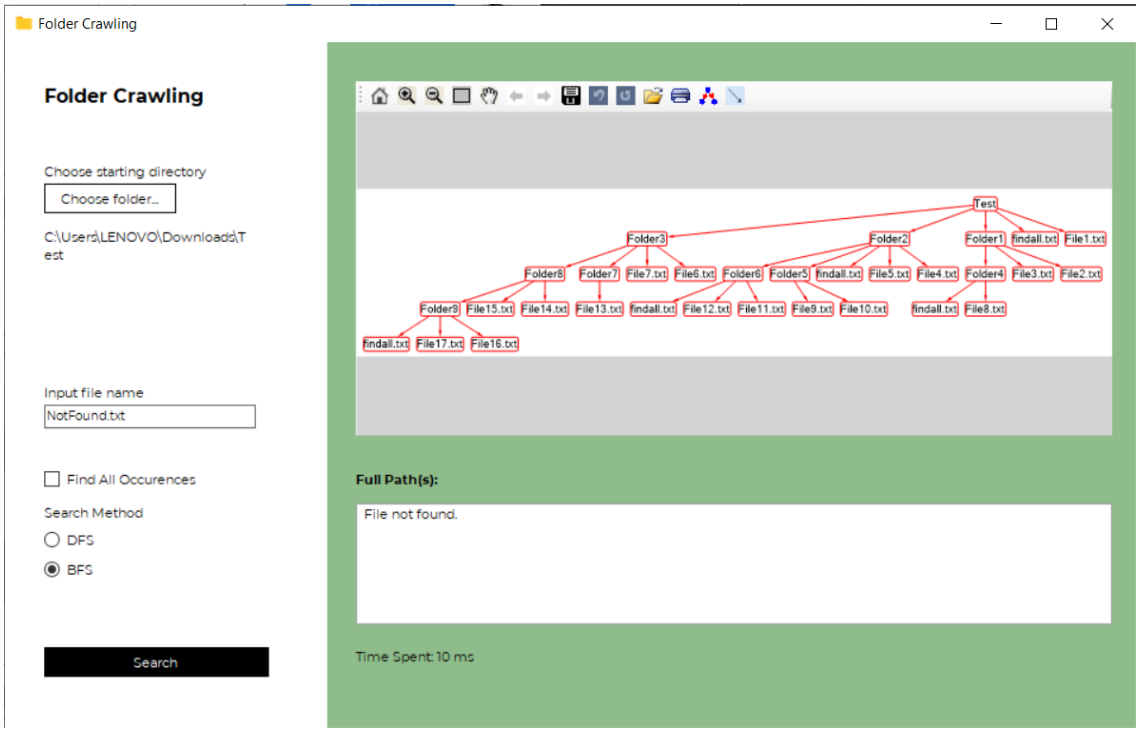
Gambar 6 Hasil Pengujian II dengan Algoritma BFS



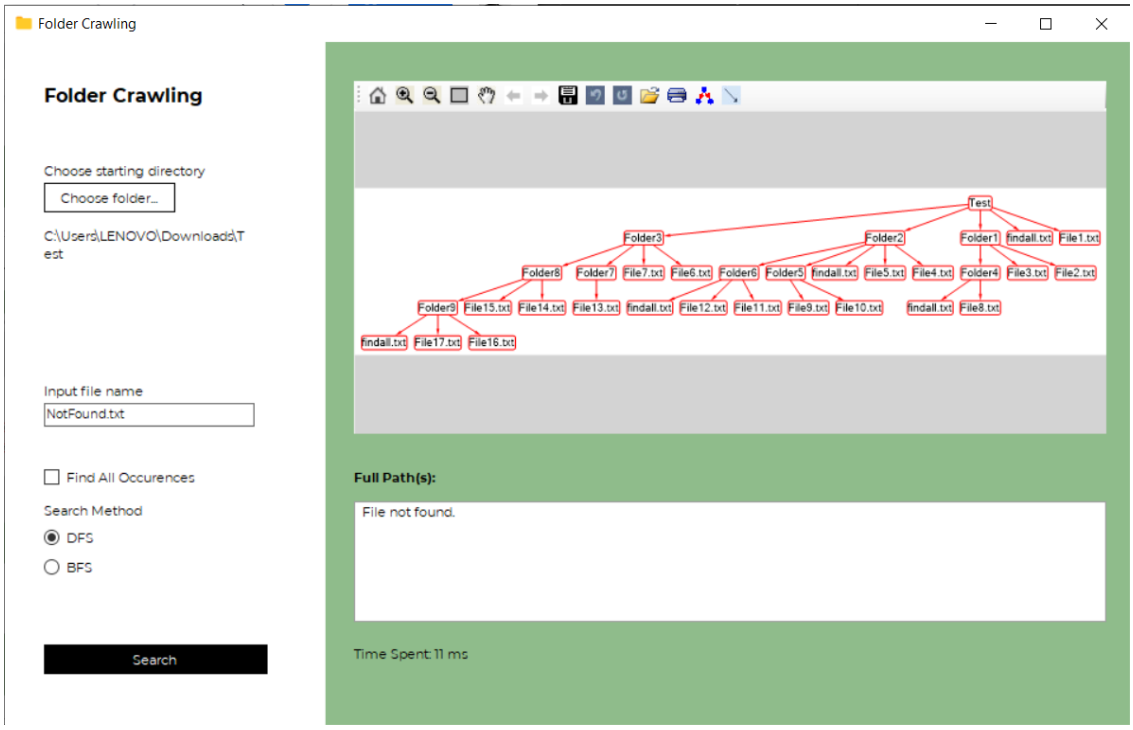
Gambar 7 Hasil Pengujian II dengan Algoritma DFS

Pada pengujian II, dilakukan pencarian semua kemunculan findall.txt pada folder Test dengan menggunakan algoritma BFS dan DFS. Dari kedua gambar, terlihat bahwa pohon yang ditampilkan serta waktu pencarian tidak memiliki perbedaan. Hal ini disebabkan karena pada pencarian semua kemunculan, perbandingan nama file dilakukan ke semua file yang ada.

c. Pengujian III



Gambar 8 Hasil Pengujian III dengan Algoritma BFS



Gambar 9 Hasil Pengujian III dengan Algoritma DFS

Pada pengujian III, dilakukan pencarian NotFound.txt pada folder Test dengan menggunakan algoritma BFS dan DFS. Dari kedua gambar, terlihat tidak ada perbedaan di antara kedua pohon Folder. Keduanya menampilkan semua file yang ada dan semua simpul berwarna merah. Hal ini disebabkan karena file NotFound.txt memang sebenarnya tidak ada pada folder Test. Ketika file yang dicari memang tidak ada, kedua algoritma mau tidak mau harus menelusuri semua simpul sampai akhir sehingga pohon yang terbentuk merupakan pohon yang lengkap seperti pada pengujian II pencarian dengan semua kemunculan.

BAB V

KESIMPULAN DAN SARAN

Kesimpulan

Dengan menggunakan Algoritma *Breadth First Search* (BFS) dan *Depth First Search* (DFS), kami telah membuat aplikasi berbasis GUI dengan menggunakan bahasa pemrograman C# dan IDEA Visual Studio. Setelah dilakukan uji coba serta studi kasus, dapat disimpulkan bahwa aplikasi yang kami buat dapat menemukan semua file yang dicari (bila memang ada) sebagaimana yang diharapkan.

Saran

Setelah menyelesaikan pengerjaan tugas besar ini, beberapa saran yang dapat kami berikan adalah lebih mempelajari bagaimana memprogram di lingkungan IDEA Visual Studio sampai benar-benar paham dan mempelajari bagaimana cara membuat GUI serta menghubungkannya dengan algoritma yang dibuat. Walaupun C# merupakan bahasa pemrograman yang baru bagi kami, tetapi sintaks-sintaksnya tidak begitu berbeda dengan Java dan C++ sehingga kendala dalam mengerjakan tugas besar ini lebih ke lingkungan memprogram di IDEA Visual Studio yang cukup rumit. Pemahaman tentang struktur folder proyek yang dibuat di Visual Studio akan mempercepat pengerjaan tugas besar kedua strategi algoritma ini. Begitu juga dengan mempelajari bagaimana cara menghubungkan GUI dengan algoritma yang sudah dibuat. Jika GUI sudah terhubung dengan algoritmanya dari awal, maka pemrogram dapat melihat input dan outputnya lebih awal sehingga proses *debugging* juga akan menjadi lebih cepat.

TAUTAN REPOSITORY GITHUB

Berikut tautan untuk mengakses *source code* yang kami simpan pada github.
https://github.com/mhelmih/FolderCrawling

TAUTAN VIDEO DEMO

Berikut tautan untuk mengakses video demo yang kami unggah di Youtube.
https://youtu.be/XD915EZn_c4

DAFTAR PUSTAKA

C# Tutorial. Javatpoint. Diakses pada 22 Maret 2022 pukul 13.43 WIB. https://www.javatpoint.com/c-sharp-tutorial

Munir, R. (2021). Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1)[PDF]. Institut Teknologi Bandung. Diakses pada 22 Maret 2022 pukul 13.43 WIB. http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf

What is .NET?. Codeacademy. Diakses pada 22 Maret 2022 pukul 13.43 WIB. https://www.codecademy.com/article/what-is-net