

LAPORAN TUGAS BESAR II

PENGAPLIKASIAN ALGORITMA BFS DAN DFS DALAM IMPLEMENTASI FOLDER CRAWLING

Laporan dibuat untuk memenuhi salah satu tugas mata kuliah

IF2211 Strategi Algoritma



Disusun oleh:

| | |
|--------------------|----------|
| Claudia | 13520076 |
| Kristo Abdi Wiguna | 13520058 |
| Jason Kanggara | 13520080 |

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

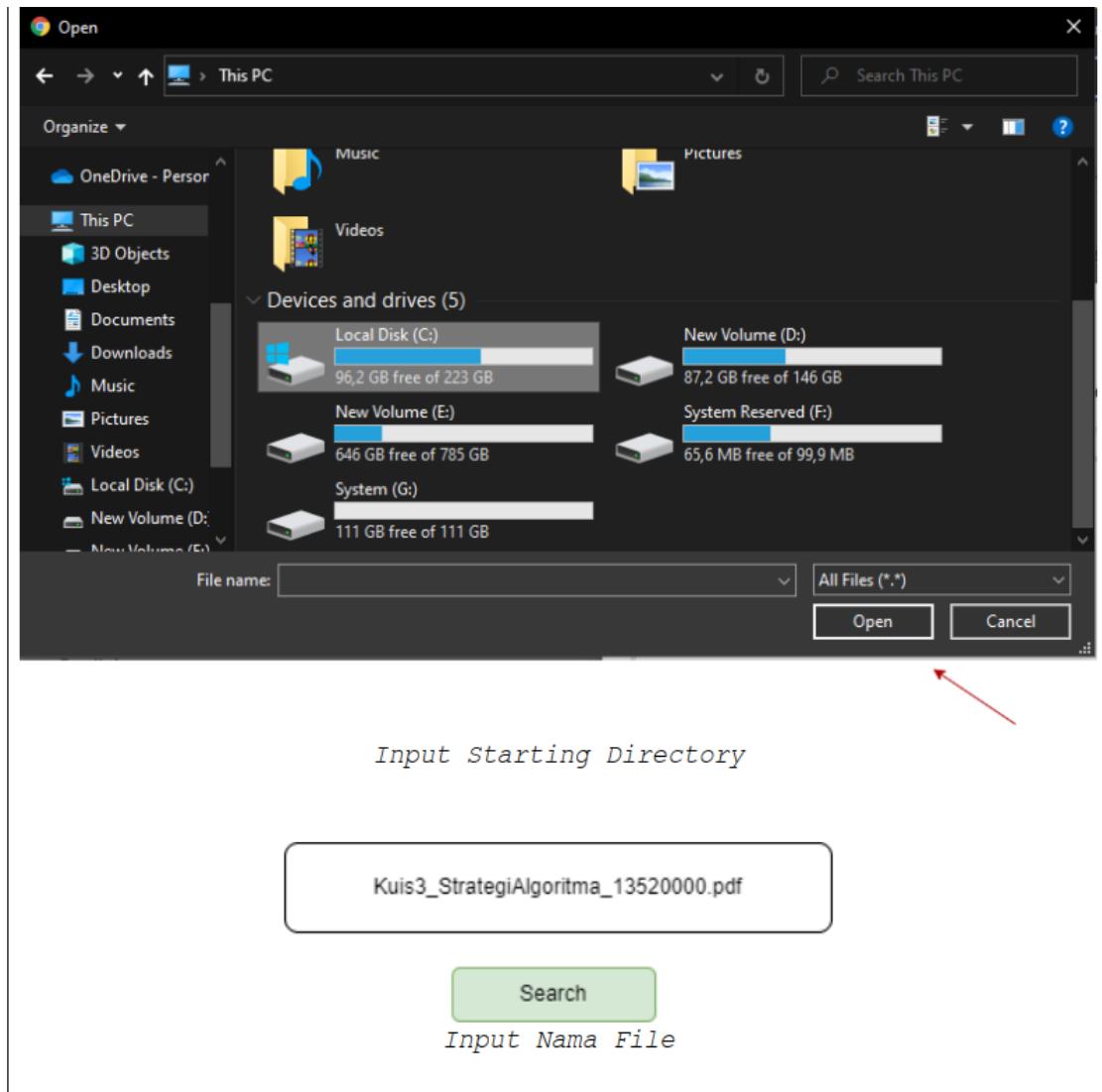
DAFTAR ISI

| | |
|---|----|
| DAFTAR ISI | 1 |
| Bab 1 Deskripsi Tugas | 2 |
| Bab 2 Landasan Teori | 5 |
| 2.1. Dasar Teori | 5 |
| 2.2 Penjelasan C# Application Development | 6 |
| Bab 3 Analisis Pemecahan Masalah | 8 |
| 3.1 Langkah Pemecahan Masalah | 8 |
| 3.2 Mapping Persoalan Algoritma BFS dan DFS | 8 |
| 3.3 Contoh Lain | 8 |
| Bab 4 Implementasi dan Pengujian | 9 |
| 4.1 Implementasi Program (Pseudocode) | 9 |
| 4.2 Struktur Data dan Spesifikasi Program | 9 |
| 4.3 Hasil Pengujian | 9 |
| 4.4 Analisis Solusi Algoritma BFS dan DFS | 9 |
| Bab 5 Kesimpulan dan Saran | 10 |
| Daftar Pustaka | 11 |

Bab 1 Deskripsi Tugas

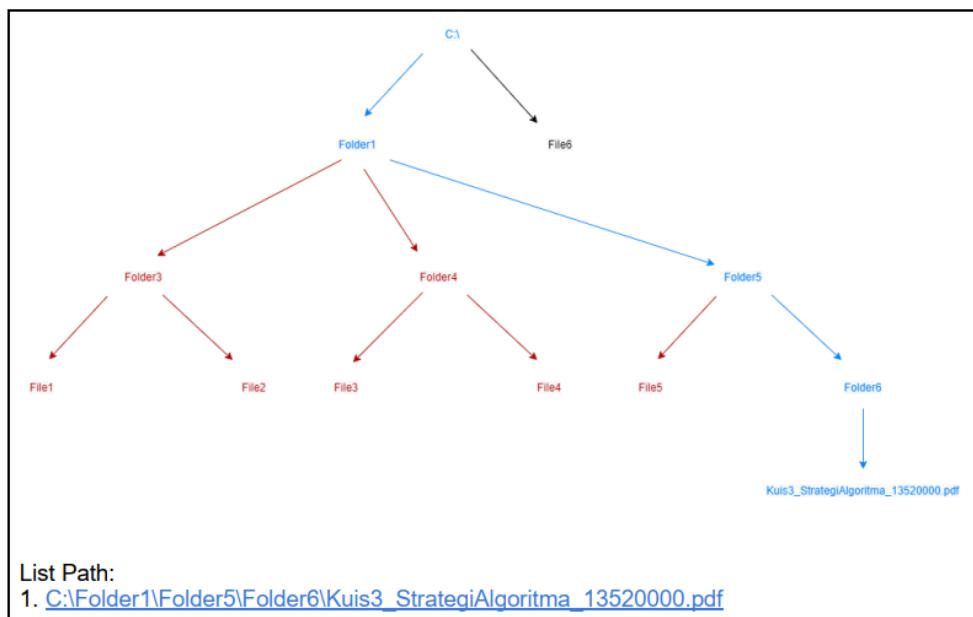
Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari file explorer pada sistem operasi, yang pada tugas ini disebut dengan Folder Crawling. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Anda dapat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



Gambar 2. Contoh input program

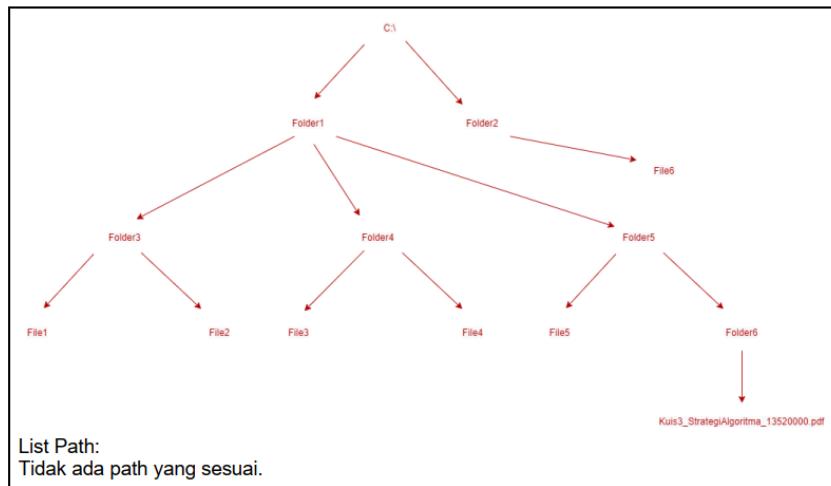
Contoh output aplikasi :



Gambar 3. Contoh output program

Misalnya pengguna ingin mengetahui langkah folder crawling untuk menemukan file Kuis3_StrategiAlgoritma_13520000.pdf. Maka, path pencarian DFS adalah sebagai berikut. C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf. Pada gambar di atas, rute yang dilewati pada pencarian DFS diwarnai dengan warna merah. Sedangkan, rute untuk menuju tempat file berada diberi warna biru. Rute yang masuk antrian tapi belum diperiksa diberi warna hitam. Anda bebas menentukan warnanya asalkan dibedakan antara ketiga hal tersebut.

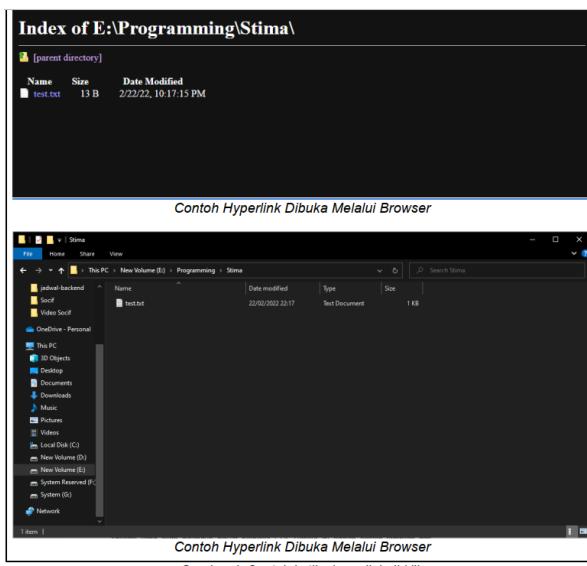
Contoh output aplikasi jika tidak ada file yang ditemukan :



Gambar 4. Contoh output program jika file tidak ditemukan

Jika file yang ingin dicari pengguna tidak ada pada direktori file, misalnya saat pengguna mencari Kuis3Probstat.pdf, maka path pencarian DFS adalah sebagai berikut: C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf → Folder6 → Folder5 → Folder1 → C:\ → Folder2 → File6. Pada gambar di atas, semua simpul dan cabang berwarna merah yang menandakan seluruh direktori sudah selesai diperiksa semua namun tidak ada yang mengarah ke tempat file berada.

Contoh hyperlink pada path :



Gambar 4. Contoh ketika hyperlink di-klik

Bab 2 Landasan Teori

2.1. Dasar Teori

Graf adalah kumpulan simpul – simpul atau *vertex* yang dihubungkan satu sama lain dengan sisi atau *edge*. Graf dapat digunakan untuk merepresentasikan objek-objek diskrit, dan juga hubungan-hubungan antara objek-objek tersebut. Graf terdiri dari simpul dan sisi. Suatu graf dapat dinyatakan sebagai $G = (V, E)$. Dimana V adalah himpunan tidak kosong dari simpul – simpul dan E adalah himpunan sisi yang menghubungkan sepasang simpul.

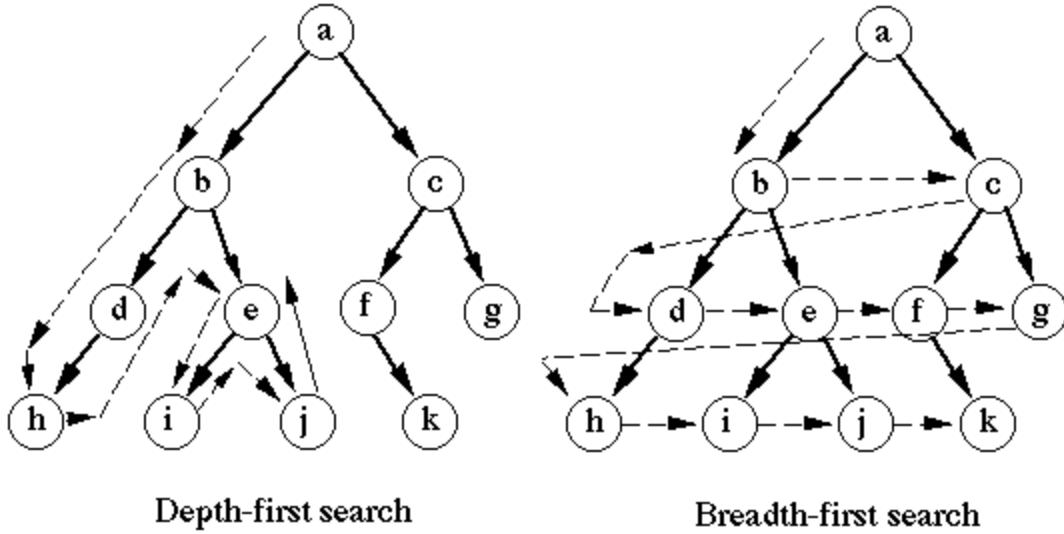
Graf digunakan untuk merepresentasikan objek – objek diskrit dan memetakan hubungan antara objek – objek tersebut. Graf dapat diklasifikasikan menjadi beberapa jenis. Berdasarkan orientasi arah pada sisinya, graf dapat digolongkan menjadi graf tak-berarah dan graf berarah. Sedangkan berdasarkan ada tidaknya kalang, graf menjadi graf sederhana dan graf tak-sederhana. Graf dinyatakan sebagai graf kosong apabila tidak ada sisi di antara simpul – simpul yang ada.

Traversal di dalam graf berarti mengunjungi simpul - simpul dengan cara sistematis (memeriksa atau memperbarui) masing masing simpul di dalam graf. Tidak seperti tree traversal, graf traversal mungkin memerlukan bahwa beberapa simpul yang dikunjungi lebih dari sekali, karena itu belum tentu diketahui sebelum transisi ke vertex yang sudah dieksplorasi. Sebagai konsekuensinya, redundansi mengunjungi simpul yang sudah pernah dikunjungi menjadi lebih umum, menyebabkan waktu komputasi meningkat.

Beberapa algoritma graf traversal yang populer adalah Breadth First Search dan Depth First Search. Breadth First Search atau singkatnya BFS adalah pencarian dari titik awal dilanjutkan ke semua cabang terurut. BFS mengunjungi simpul cabang terlebih dahulu daripada anak dari cabang dan queue dapat digunakan sebagai urutan simpul pencarian. Algoritma ini sering digunakan untuk mencari jarak terdekat dari satu simpul ke simpul lainnya.

Depth first search adalah algoritma graf traversal yang berbeda dari BFS dan DFS mengunjungi anak cabang terlebih dahulu dibandingkan simpul saudara cabang. DFS

menelusuri jalur hingga dalam dan melakukan backtrack ketika tidak ada simpul anak yang bisa dikunjungi lagi. Biasanya menggunakan stack seolah untuk menyimpan urutan yang perlu dicari dengan menambah di atas stack terus.



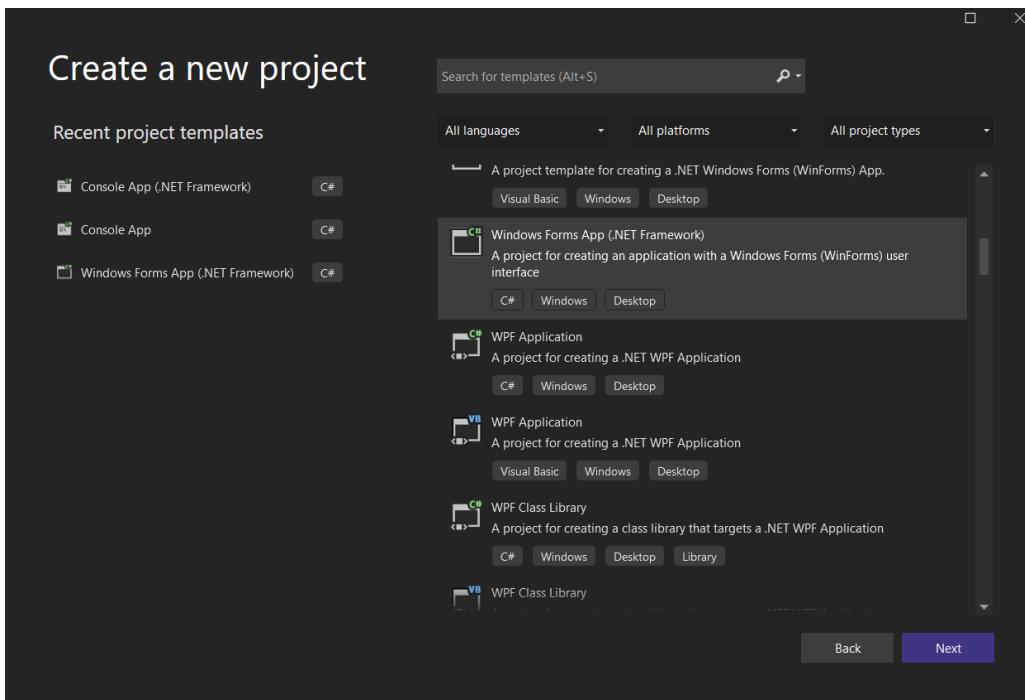
Depth-first search

Breadth-first search

Sumber : https://medium.com/@tim_ng/bfs-and-dfs-52d3cb642a0e

2.2 Penjelasan C# Application Development

C# adalah sebuah bahasa pemrograman berorientasi objek yang dikembangkan oleh Microsoft. C# memiliki banyak fitur yang juga ada pada bahasa C++ dan Java seperti *collection*, *reflection*, *multiple inheritance* dan lain-lain. Bahasa C# bersifat general-purpose yaitu bisa digunakan untuk membuat software komputer, aplikasi desktop, ataupun web, biasanya C# juga digunakan untuk membuat games. C# merupakan bahasa yang mudah dimengerti dan menyediakan berbagai library yang bermanfaat. Untuk menjalankan C#, dibutuhkan IDE dan Framework. IDE dan Framework yang digunakan yaitu Visual Studio dan .NET. .NET adalah sebuah framework yang digunakan untuk mempermudah pengembangan aplikasi berbasis Windows. Keuntungan dari framework .NET adalah open-source sehingga dapat digunakan secara gratis, framework .NET juga bisa digunakan pada bahasa pemrograman selain C# seperti VB.Net, J++, dan lain-lain.



Ada banyak jenis aplikasi yang disediakan oleh Visual Studio seperti Windows Form Application, Console Application, dan lain-lain namun pada tugas ini kami mengimplementasikannya menggunakan Windows Forms Application untuk membuat Graphical User Interface (GUI).

Bab 3 Analisis Pemecahan Masalah

3.1 Langkah Pemecahan Masalah

Secara garis besar, pemecahan masalah dapat dilakukan mulai dari pembuatan fungsi DFS dan BFS, memvisualisasikan pencarian ke dalam graph, dan pembuatan desktop application untuk program tersebut.

1. Penerimaan masukan pengguna, folder root yang dipilih untuk dilakukannya pencarian file. Lalu, penerimaan nama file beserta ekstensi misalnya “prak1.sql” sebagai nama file yang dicari. Penerimaan checklist apakah pengguna ingin mencari semua keberadaan file dengan nama yang sama atau cukup satu file yang ditemukan terlebih dahulu sesuai dengan algoritma searching yang dipilih.
2. Penerimaan masukan pengguna dengan memilih metode pencarian DFS atau BFS.
3. Melakukan pencarian berdasarkan masukan pengguna hingga menemukan hasilnya.
4. Jika file ditemukan, menampilkan path menuju file tersebut dan waktu yang dibutuhkan untuk mencari file tersebut.
5. Jika file tidak ditemukan, menampilkan keluaran bahwa “File not found”.
6. 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 beserta file yang merupakan rute hasil pertemuan.

3.2 Mapping Persoalan Algoritma BFS dan DFS

Pada persoalan ini, vertex atau node dari sebuah graf merepresentasikan folder atau file sedangkan sisi dari sebuah graf merepresentasikan hubungan subfolder atau hubungan file di dalam sebuah folder.

Dengan Algoritma BFS: Penelusuran file dimulai pada starting directory, akan dilakukan pengecekan terhadap subfolder dan file yang ada pada directory tersebut. Subfolder pada sebuah directory akan dimasukkan pada antrian subfolder yang akan dicek. Jika findAllOccurrence bernilai false, maka pencarian akan diberhentikan jika file sudah ditemukan pertama kalinya. Jika true, maka proses pencarian akan dilanjutkan dengan memasuki salah

satu subfolder, jika tidak ditemukan akan dilakukan pencarian pada subfolder pada antrian lain sampai semua subfolder telah dilakukan pengecekan.

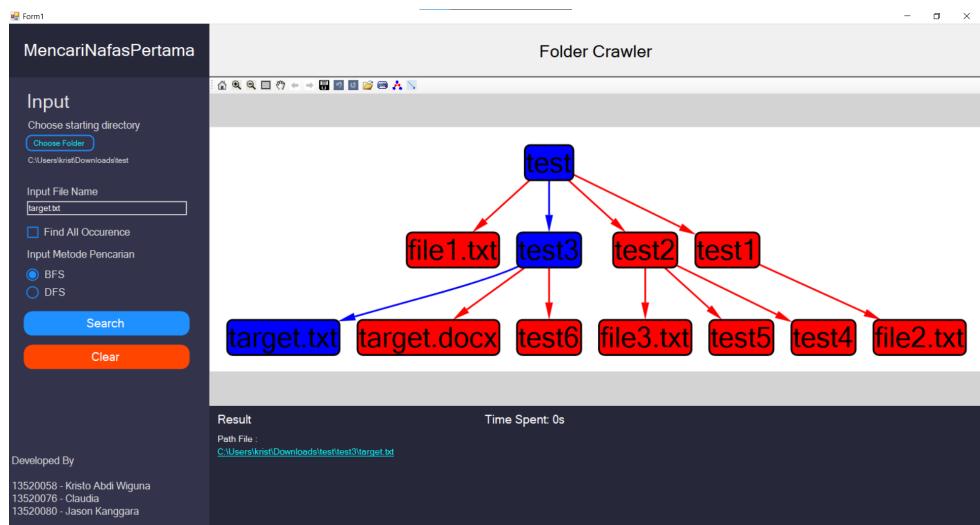
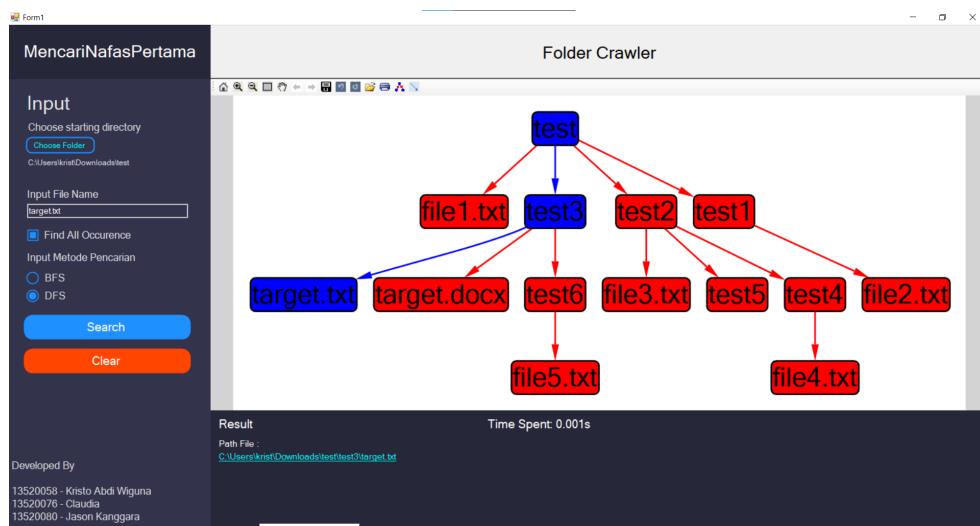
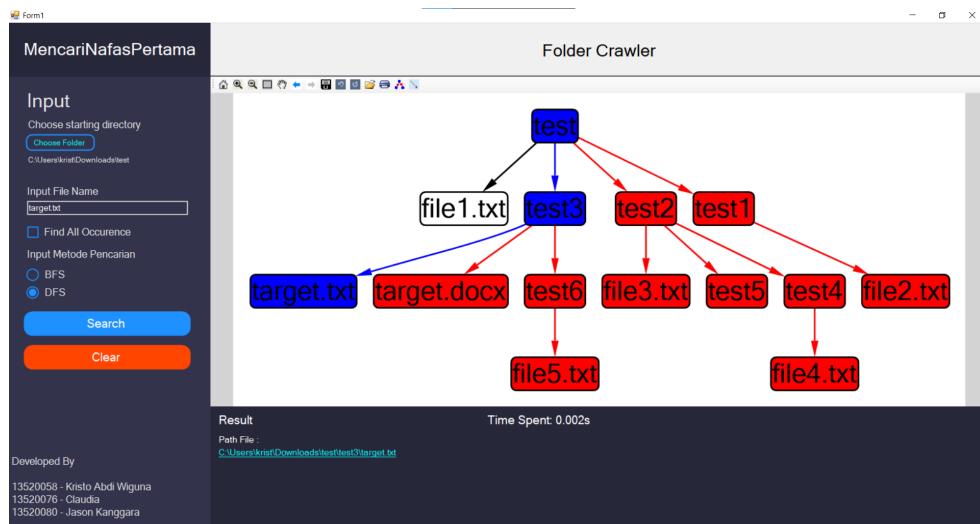
Dengan Algoritma DFS: Penelusuran file dimulai pada starting directory, akan dilakukan pengecekan terhadap subfolder dan file yang ada pada directory tersebut. Jika findAllOccurrence bernilai false, maka pencarian akan diberhentikan jika file sudah ditemukan pertama kalinya. Jika tidak, maka proses pencarian akan dilanjutkan dengan memasuki salah satu subfolder dan mengulangi proses rekursif tersebut hingga menelusuri semua subdirektori dan mengevaluasi semua file yang ada sehingga dengan pasti semua subdirektori dan file telah ditelusuri sehingga tidak akan vertex atau sisi yang menghubungkan node yang berwarna hitam dan hanya merah atau biru. Proses pencarian dilakukan hingga terdalam dan tidak ada subdirektori lagi yang bisa dicapai, ketika tidak ada subdirektori lagi, maka program akan melakukan backtrack ke parent node untuk mencari subdirektori yang belum dikunjungi.

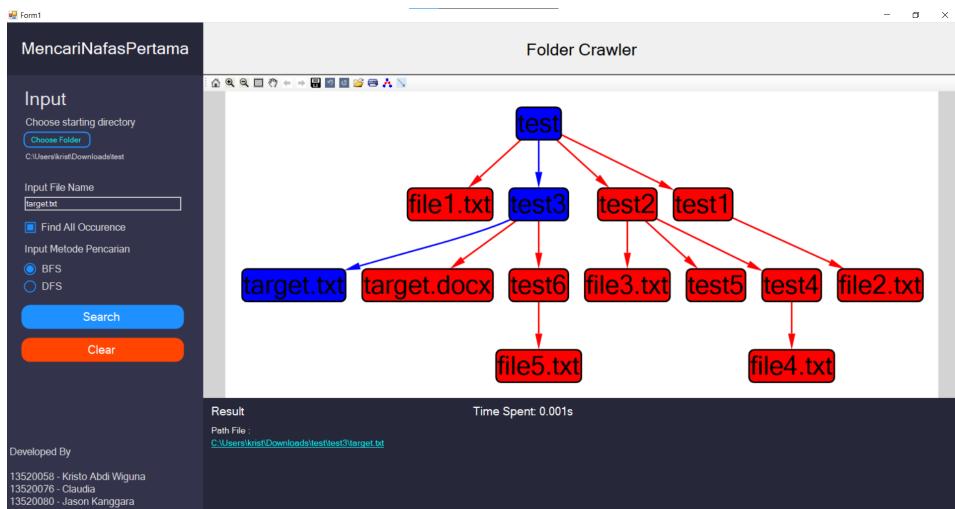
3.3 Contoh Ilustrasi Kasus Lain

1. Kasus file target dengan nama file sama, ekstensi file berbeda di folder yang sama

Contoh : target (target.txt dan target.docx)

| C > Downloads > test > test3 > | | | | |
|--------------------------------|--------------------|---------------------|------|--|
| Name | Date modified | Type | Size | |
| test6 | 3/18/2022 10:46 AM | File folder | | |
| target.docx | 3/21/2022 11:11 AM | Microsoft Word D... | 0 KB | |
| target.txt | 3/19/2022 1:31 AM | Text Source File | 0 KB | |

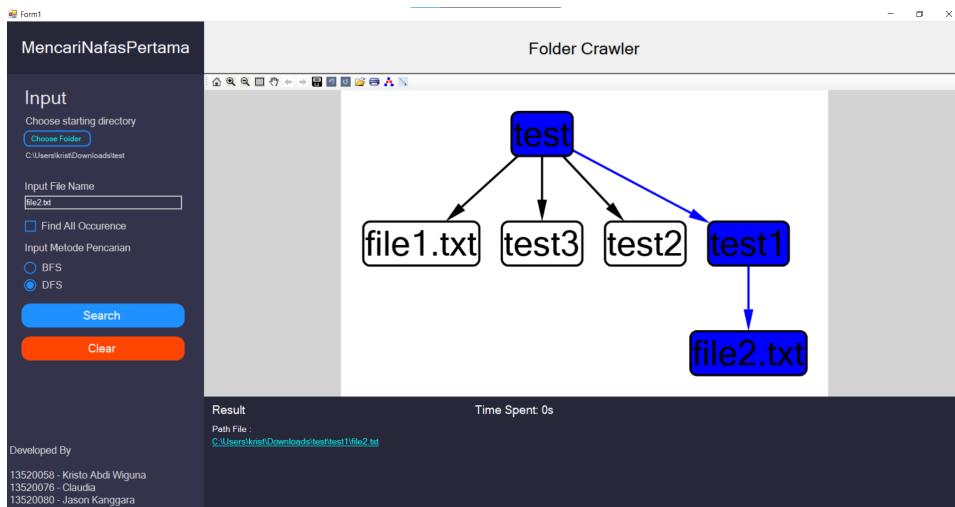


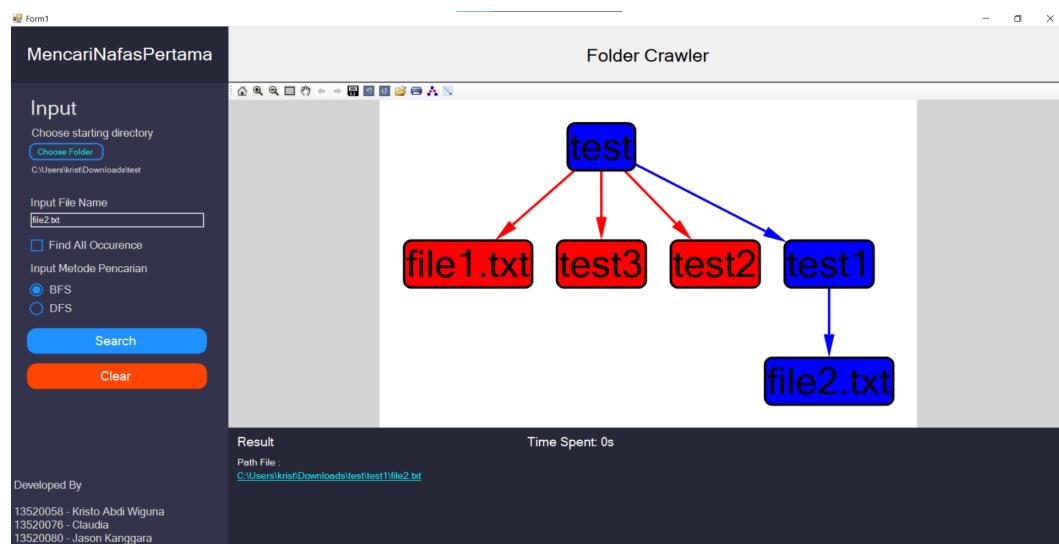
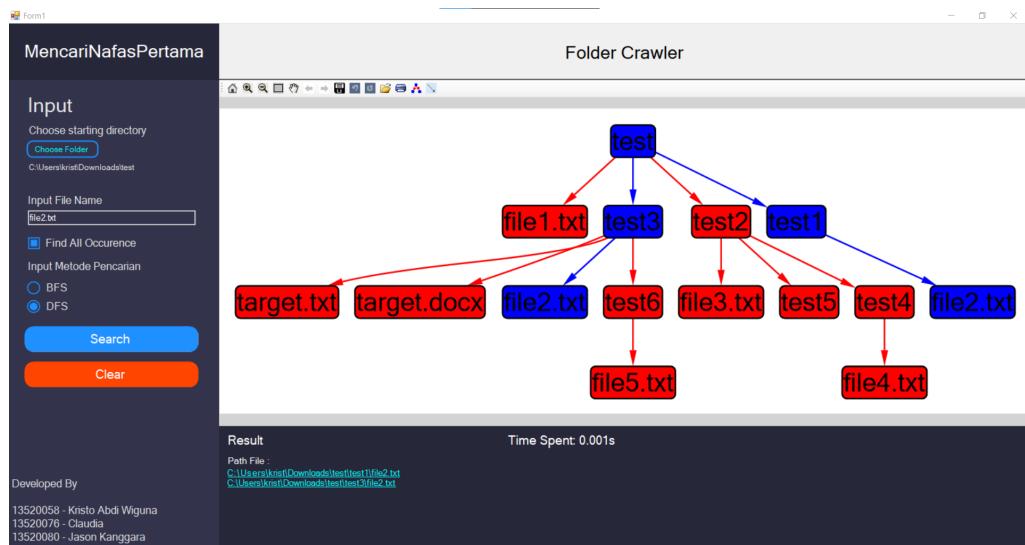


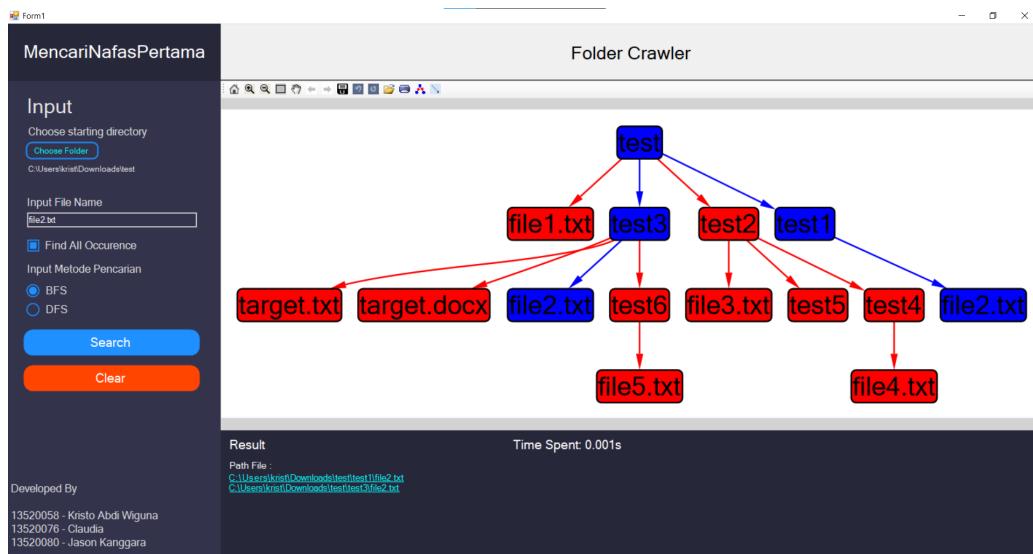
Terlihat bahwa file yang dicari berbeda dengan nama file yang sama karena ekstensi file yang berbeda.

2. Kasus file target dengan nama sama di folder yang berbeda

Contoh : target (test\test1\file2.txt dan test\test3\file2.txt)







Algoritma BFS dan DFS juga dapat digunakan dalam kehidupan sehari - sehari, misalnya yaitu navigasi GPS yang mencari rute terdekat untuk mencapai tujuan dengan estimasi waktu yang telah disediakan oleh GPS. Selain itu, algoritma BFS dan DFS juga dapat digunakan untuk search-engine crawler, labirin atau sudoku, dan lain - lain.

Bab 4 Implementasi dan Pengujian

4.1 Implementasi Program (Pseudocode)

```
procedure BFS(input root : string, input target: string, input
findAll : boolean)
KAMUS :
    listFilesAndDirectory : List of String
    listChildFilesAndDirectory : List of String
    toVisitQueue : Queue of String
ALGORITHM :
    listFilesAndDirectory <- {}
    AddFiles(root, listFilesAndDirectory)
    foreach filedir in listFilesAndDirectory :
        listFileDirectory <- {root, filedir}
        Add listFileDirectory in listFilesAndDirectory
    endfor

    if (listFilesAndDirectory is not empty) then
        toVisitQueue <- {}
        foreach filedir in listFilesAndDirectory :
            listFileDirectory <- {root, filedir}
            Add listFileDirectory in adjacencyList
            if (filedir is a Directory) then
                toVisitQueue <- filedir
            else if (filedir is a File) then
                if (filedir is target) then
                    Add filedir in fileLocationResult
                    if ( findAll = false ) then
                        return
                    endif
                endif
            endif
        endforeach
    endif

    while ( count(toVisitQueue) > 0 ) do
        currentDirectory <- Dequeue(toVisitQueue)
        listChildFilesAndDirectory <- {}
        foreach child in listChildFilesAndDirectory :
            listChildFileDirectory <- {currentDirectory,
child}
            Add listChildFileDirectory in adjacencyList
```

```

        if (child is a Directory) then
            toVisitQueue <- filedir
        else if (child is a File) then
            if (filedir is target) then
                Add child in fileLocationResult
                if ( findAll = false ) then
                    return
                endif
            endif
        endif
    endfor
endif

procedure DFS(input root : string, input target: string, input
findAll : boolean)
KAMUS :
    listFilesAndDirectory : List of String
ALGORITHM :
    listFilesAndDirectory <- {}
    AddFiles(root, listFilesAndDirectory)
    foreach filedir in listFilesAndDirectory :
        listFileDirectory <- {root, filedir}
        Add listFileDirectory in listFilesAndDirectory
    endfor

    if (listFilesAndDirectory is not empty) then
        foreach filedir in listFilesAndDirectory :
            if (fileFound = false and findAll = false) then
                listFileDirectory <- {root, filedir}
                Add listFileDirectory in adjacencyList
            else if (findAll = true) then
                listFileDirectory <- {root, filedir}
                Add listFileDirectory in adjacencyList
            endif

            if (filedir is a File) then
                if (filedir is target) then
                    Add filedir in fileLocationResult
                    fileFound <- true
                endif
            else if (filedir is a Directory) then
                DFS(filedir, target, findAll)

```

```

        if (findAll = false and fileFound = true) then
            return
        endif
    endif
endfor
endif

procedure AddFiles(input root : string, input/output
listFilesAndDirectory : List of String)
KAMUS
    filesInFolder, folder : List of String
ALGORITHM
    try :
        folder <- GetDirectoriesIn(root)
        filesInFolder <- GetFilesIn(root)
        foreach filedir in filesInFolder
            Add filedir in folder
        endfor
        listFilesAndDirectory <- folder
    catch (exception) :
        // do nothing

```

4.2 Struktur Data dan Spesifikasi Program

```

Form1

fileLocationResult : List<string> -> menyimpan lokasi file
setelah dilakukan pencarian
adjacencyList : List<List<string>> -> menyimpan hubungan
antara folder dengan file
locationList : List<List<string>> -> menyimpan path file
target
unvisitedList : List<List<string>> -> menyimpan list of
unvisited directory atau file yang sedang mengantri
findAllOccurrence : boolean -> menyimpan boolean yang bernilai
true jika semua file target akan dicari dalam sebuah
directory, false jika hanya dicari kemunculan pertama

```

| |
|---|
| searchMethod : string -> menyimpan nama algoritma yang digunakan yaitu bfs atau dfs startDirectory : string -> menyimpan nama starting directory fileName : string -> menyimpan nama file target execTime : float -> menyimpan waktu yang dibutuhkan untuk mencari file target fileFound : boolean -> boolean yang menentukan apakah file target sudah ditemukan atau tidak |
|---|

| |
|-----|
| BFS |
|-----|

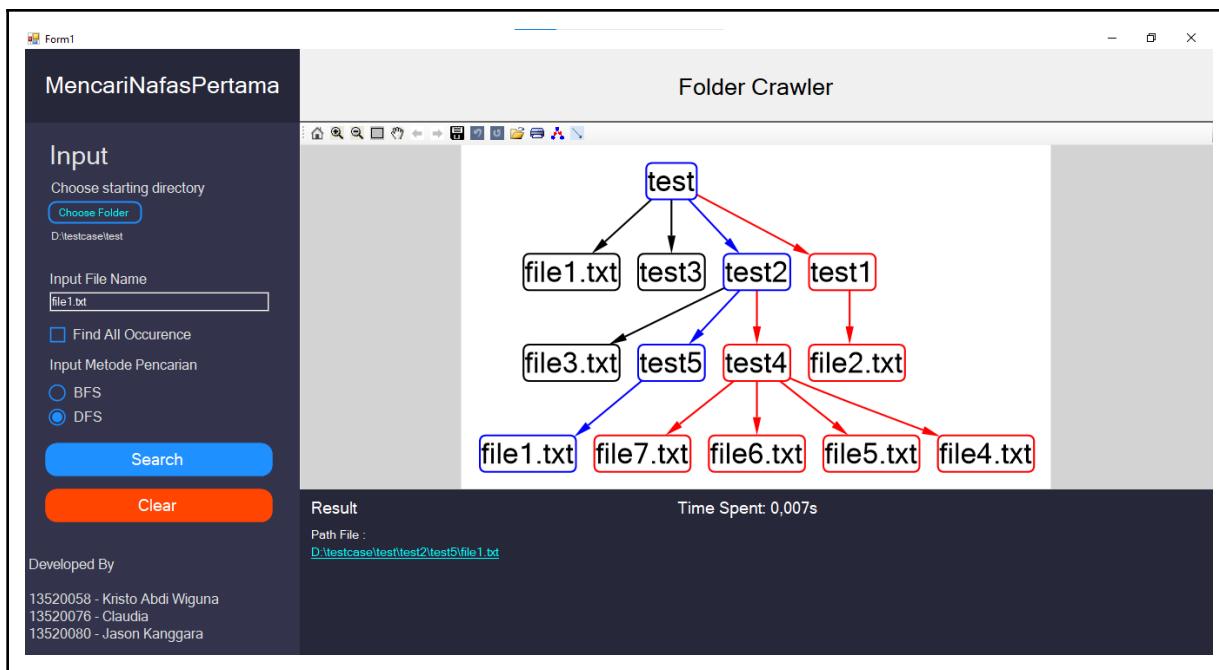
| |
|---|
| listFilesAndDirectory : List<string> -> menyimpan nama file dan subfolder pada sebuah directory toVisitQueue : Queue<string> -> menyimpan nodes yang akan dikunjungi |
|---|

| |
|-----|
| DFS |
|-----|

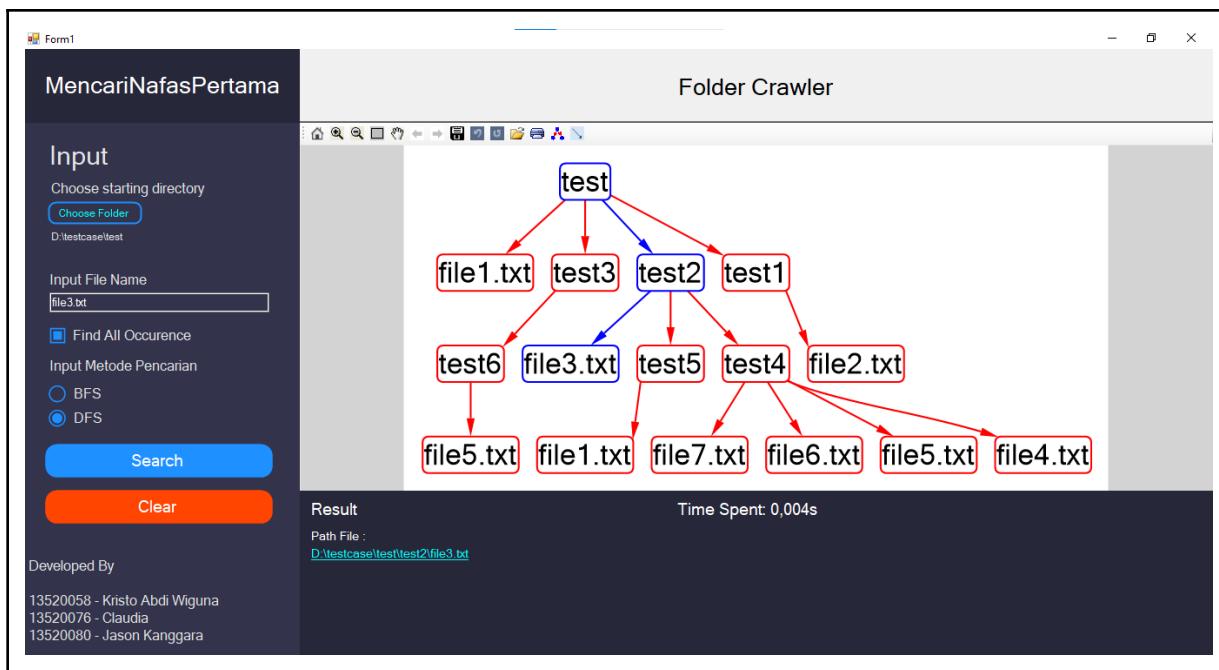
| |
|---|
| listFilesAndDirectory : List<string> -> menyimpan nama file dan subfolder pada sebuah directory |
|---|

4.3 Hasil Pengujian

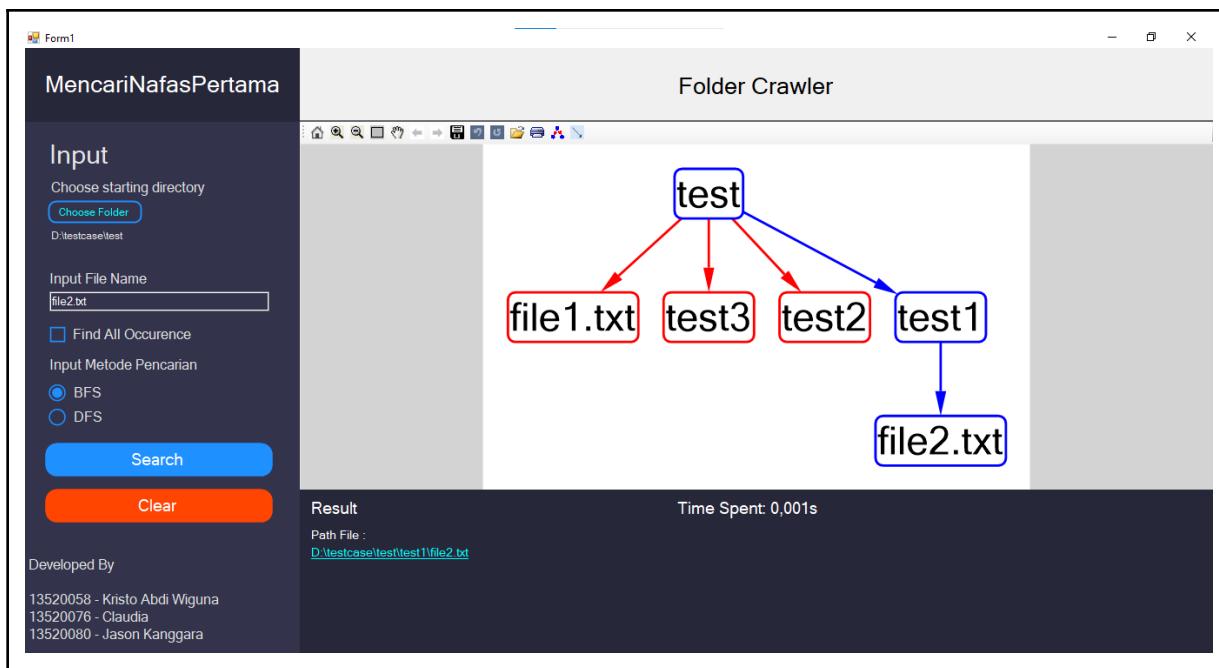
| Test 1 | |
|----------------------------|-------------------|
| Folder Asal | D:\ testcase\test |
| File Tujuan | file1.txt |
| Metode Pencarian | DFS |
| <i>Find All Occurrence</i> | False |
| Hasil Pengujian | |



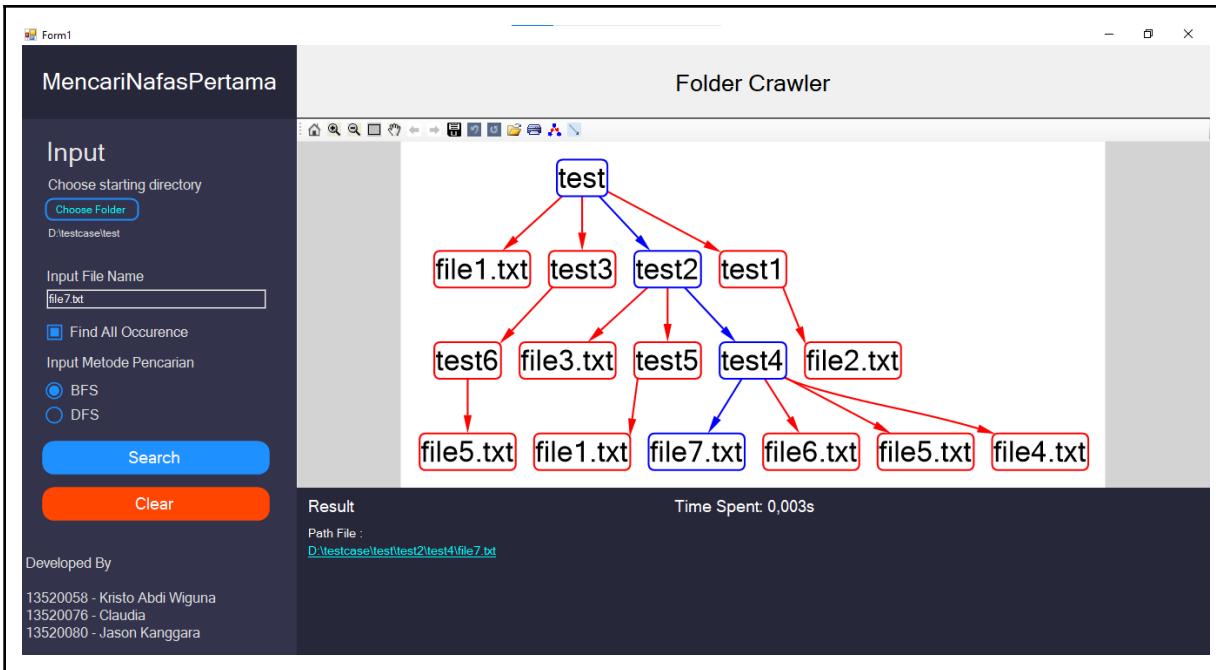
| Test 2 | |
|----------------------------|-------------------|
| Folder Asal | D:\ testcase\test |
| File Tujuan | file3.txt |
| Metode Pencarian | DFS |
| <i>Find All Occurrence</i> | True |
| Hasil Pengujian | |



| Test 3 | |
|----------------------------|------------------|
| Folder Asal | D:\testcase\test |
| File Tujuan | file2.txt |
| Metode Pencarian | BFS |
| <i>Find All Occurrence</i> | False |
| Hasil Pengujian | |



| Test 4 | |
|----------------------------|-------------------|
| Folder Asal | D:\ testcase\test |
| File Tujuan | file7.txt |
| Metode Pencarian | BFS |
| <i>Find All Occurrence</i> | True |
| Hasil Pengujian | |



4.4 Analisis Solusi Algoritma BFS dan DFS

Perbedaan yang paling mendasar antara algoritma BFS dan DFS adalah proses pencariannya yang sesuai dengan penamaan dari kedua algoritma tersebut. BFS, atau *Breadth First Search*, adalah algoritma pencarian yang mengutamakan pencarian pada masing - masing tingkatan terlebih dahulu. Jika diterapkan pada graf pohon, dengan algoritma BFS, proses dilakukan dengan menelusuri setiap node yang terdapat di tingkat satu, lalu dilanjutkan ke tingkat dua, tiga, dan seterusnya, sampai semua node ditelusuri, atau sudah menemukan solusinya. Untuk algoritma DFS, atau *Depth First Search*, adalah algoritma pencarian yang mengutamakan pencarian kedalaman node yang masih dapat ditelusuri, jika sudah sampai node terdalam dan belum mendapatkan solusinya, maka akan dilakukan *backtracking* ke node sebelumnya untuk melanjutkan pencarian kedalaman pada node dalam antrian. Algoritma BFS menganut konsep *Queue*, yaitu *First In First Out*, dan DFS menganut konsep *Stack*, yaitu *Last In First Out*. Kasus dimana BFS dan DFS mengungguli atau diungguli tergantung dengan bagaimana bentuk dari kasus tersebut dan solusi dari kasus tersebut.

Untuk mengamati kapan BFS lebih unggul atau DFS lebih unggul, akan digunakan kasus pada tugas besar ini, yaitu melakukan pencarian suatu file dengan algoritma BFS dan DFS. Pada Test 1 misalnya, melakukan pencarian “file1.txt” pada suatu folder dengan algoritma DFS. Terlihat pada graf, dengan algoritma DFS, proses pencarian harus menelusuri sampai node terdalam tidak berdasarkan masing masing lapisan, sehingga file yang ditemukan berada di dalam folder test2/test5, padahal, jika dilihat dari graf, terdapat “file1.txt” pada lapisan/tingkatan pertama. Jika menggunakan algoritma BFS, maka “file1.txt” dapat ditemukan lebih cepat karena dengan algoritma BFS, akan memeriksa setiap node yang ada pada setiap lapisan folder. Pada kasus ini, algoritma BFS mengungguli algoritma DFS dalam pencarian file dalam folder.

Untuk kasus DFS mengungguli BFS, dapat kita lihat pada pengujian Test 3, yaitu mencari file “file2.txt”. Pada hasil pengujian di atas, dilakukan pencarian “file2.txt” dengan algoritma BFS, sehingga proses pencarian harus memeriksa setiap node yang ada pada lapisan pertama, lalu lanjut ke lapisan ke dua dan file ditemukan. Jika kita menggunakan algoritma DFS, maka pencarian file tersebut lebih cepat, karena dengan DFS akan diperiksa secara mendalam pada simpul yang diperiksa pertama kali, yaitu folder test1, dilanjutkan ke simpul yang berada di dalam folder test1, yaitu “file2.txt” yang merupakan jawabannya, sehingga simpul lain yang setingkat dengan folder test1 tidak diperiksa karena DFS tidak memeriksa setiap simpul yang ada pada suatu lapisan. Oleh karena itu pada kasus Test 3, algoritma DFS lebih cepat dibandingkan dengan algoritma BFS.

Bab 5 Kesimpulan dan Saran

Kesimpulan

Berdasarkan yang sudah dipaparkan, dapat diambil kesimpulan bahwa implementasi algoritma BFS dan DFS dapat digunakan untuk kasus - kasus dunia nyata misalnya untuk *folder crawling*. Pada tugas implementasi BFS dan DFS untuk *folder crawling* dengan *desktop application* ini dapat berjalan dengan baik. Metode pencarian DFS dan BFS pada kasus *folder crawling* dapat menemukan solusi yang baik dan tidak jauh berbeda. Namun, karena implementasi DFS dan BFS yang berbeda, terdapat perbedaan cara penelusuran node, maka jalur yang dilalui saat proses traversal berbeda dan metode BFS cenderung menghasilkan solusi jalur yang lebih pendek dan lebar sedangkan DFS cenderung menghasilkan solusi jalur yang lebih panjang dan sempit. Dalam hal ini, program kami sudah dapat menerapkan metode pencarian BFS dan DFS dalam *folder crawling* dengan media *desktop application* dengan tepat.

Saran

Algoritma struktur data BFS dan DFS yang digunakan pada tugas besar kali ini dirasa masih kurang efisien. Masih banyak struktur data yang redundant dan seharusnya bisa dikurangi. Begitu juga dengan algoritma DFS dan BFS yang mungkin bisa lebih memanfaatkan paradigma pemrograman berorientasi objek. Algoritma dapat ditingkatkan dengan mengurangi memori yang dipakai, seperti penyederhanaan struktur data yang dipakai dan meminimalisir pembuatan variabel baru. GUI dapat dibuat menjadi lebih estetik dengan memberikan tampilan yang lebih menarik, seperti melakukan desain background image maupun penggunaan warna yang sinkron dan indah dipandang.

Link Video

https://youtu.be/jEj9_pbuPy0

Link Repository

<https://github.com/kristabdi/graph-folder-crawling>

Daftar Pustaka

[1] Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2022), Breadth/Depth First Search (BFS/DFS) Bagian 1. Diakses online dari
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>
pada 15 Maret 2022.

[2] Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2022), Breadth/Depth First Search (BFS/DFS) Bagian 2. Diakses online dari
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>
pada 20 Maret 2022.

[3] Microsoft. (2022). C# Programming Guide. Diakses online dari
<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/> pada 14 Maret 2022.

[4] Microsoft (2022), MS Automatic Graph Layout. Diakses online dari
<https://github.com/microsoft/automatic-graph-layout> pada 19 Maret 2022.