Tugas Besar IF2211 Strategi Algoritma: Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling

Anggota Kelompok :

Aditya Prawira N	13520049
Dimas Shidqi P	13520087
Maria Khelli	13520115

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

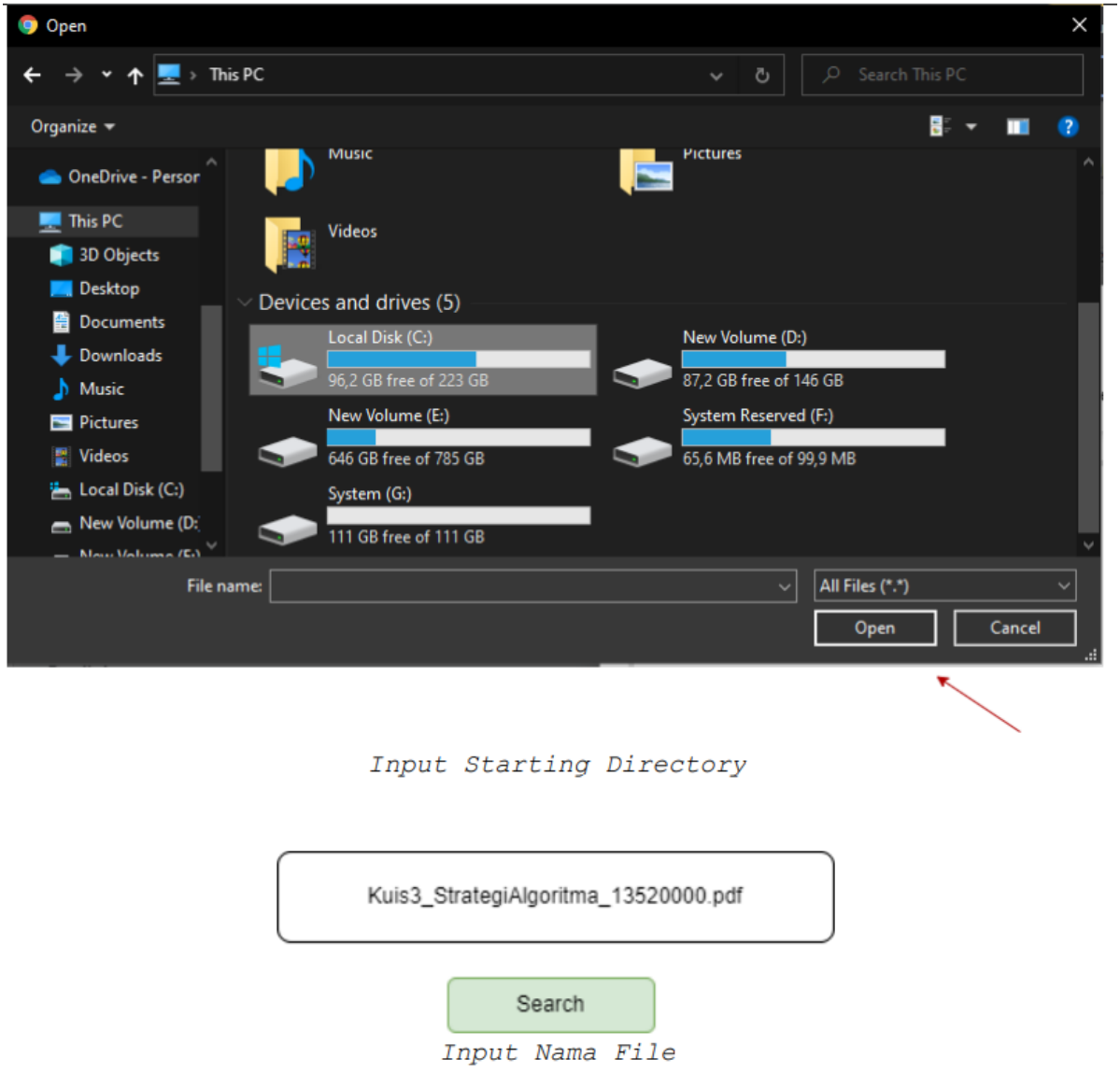
DAFTAR ISI

BAB 1
DESKRIPSI TUGAS

Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari file explorer pada sistem operasi, yang pada tugas ini disebut dengan Folder Crawling. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Anda dapat menelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang Anda inginkan. Anda juga diminta untuk memvisualisasikan hasil dari pencarian folder tersebut dalam bentuk pohon.

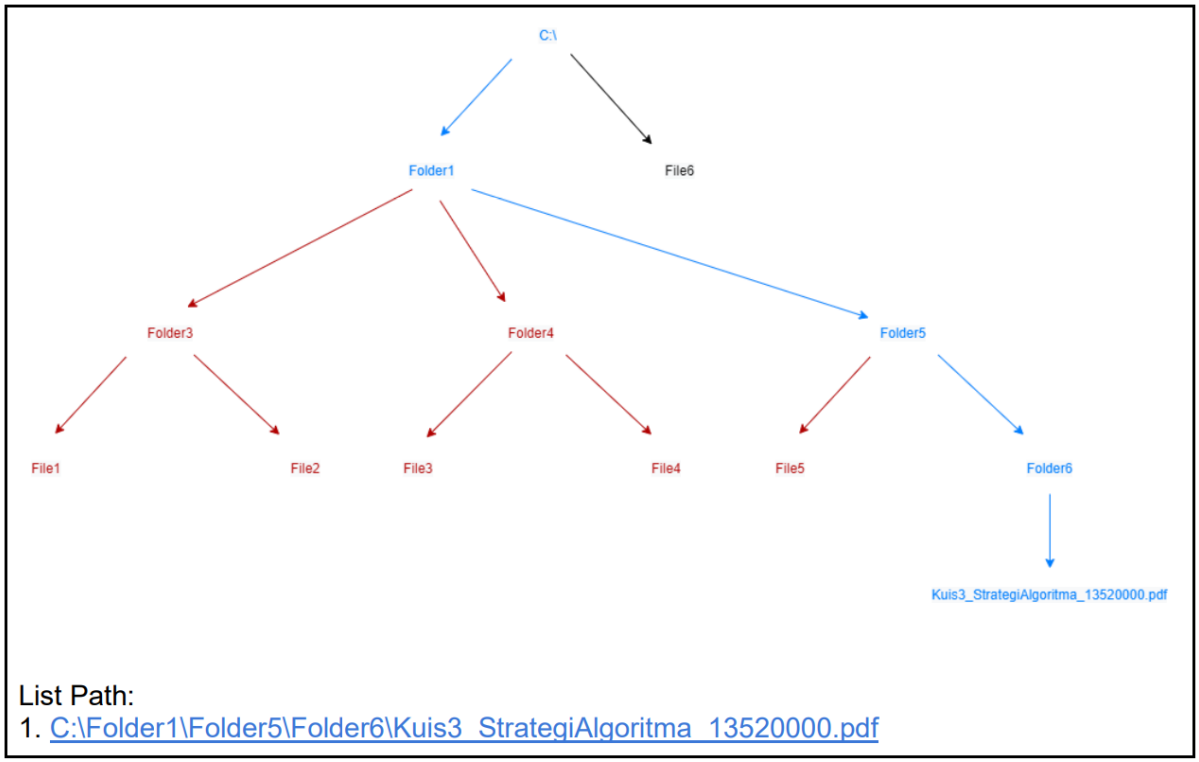
Selain pohon, Anda diminta juga menampilkan list path dari daun-daun yang bersesuaian dengan hasil pencarian. Path tersebut diharuskan memiliki hyperlink menuju folder parent dari file yang dicari, agar file langsung dapat diakses melalui browser atau file explorer. Contoh hal-hal yang dimaksud akan dijelaskan di bawah ini.

Contoh masukan aplikasi:



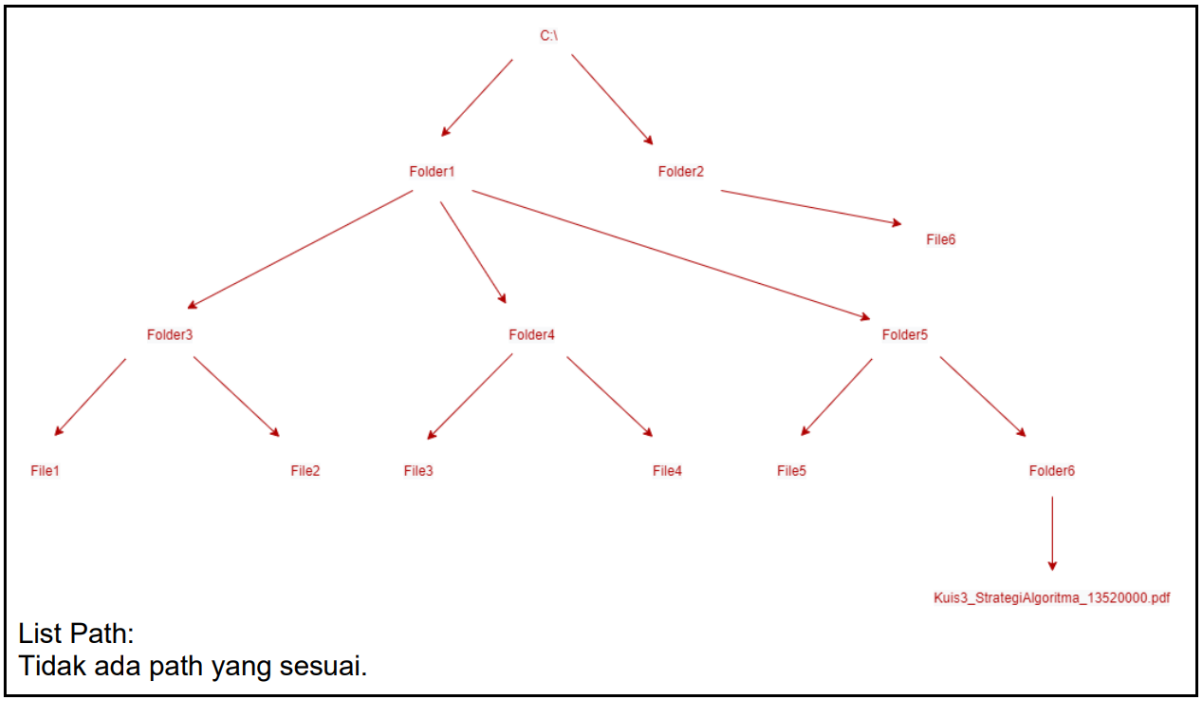
Gambar 1 Contoh input program

Contoh keluaran aplikasi:



Gambar 2 Contoh keluaran program

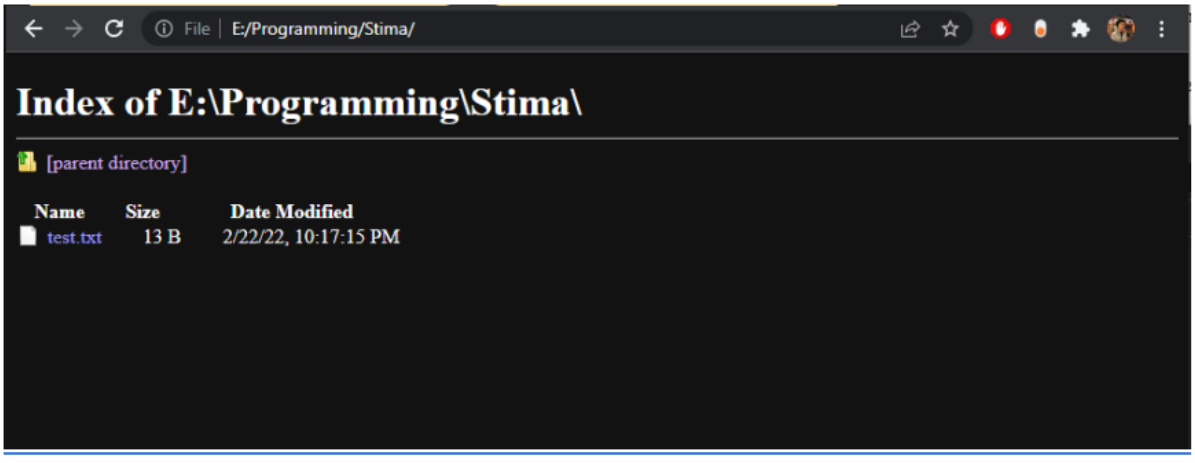
Misalnya pengguna ingin mengetahui langkah folder crawling untuk menemukan file Kuis3_StrategiAlgoritma_13520000.pdf. Maka, path pencarian DFS adalah sebagai berikut. C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → 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.



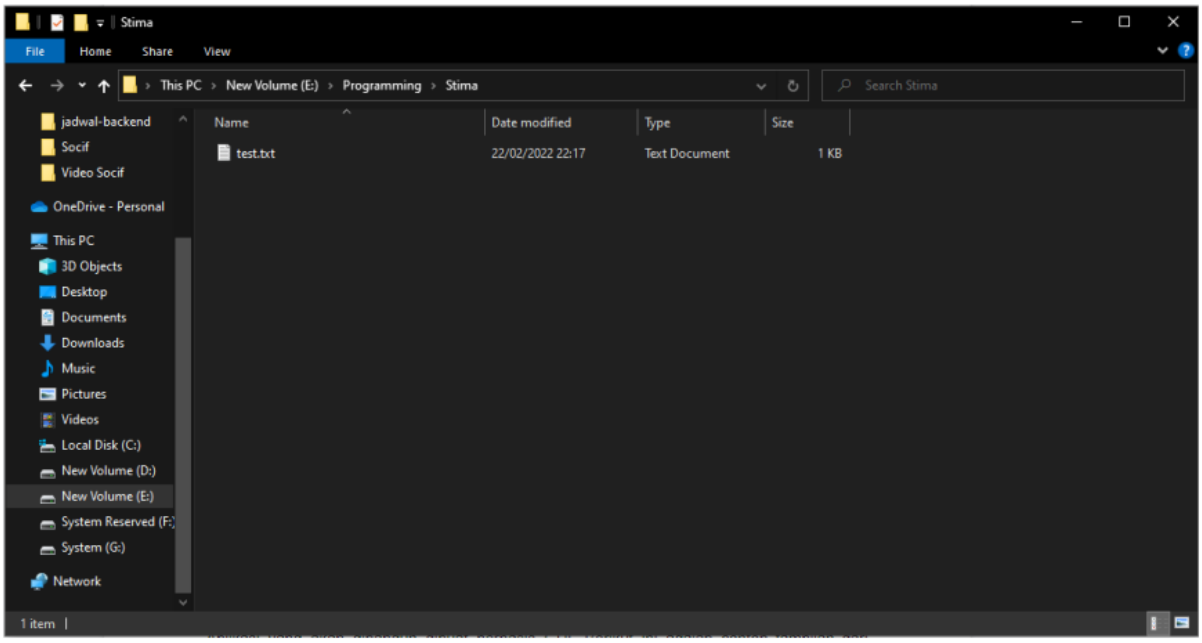
Gambar 3 Contoh keluaran program jika file tidak ditemukan

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 hyperlink dibuka melalui broswer



Gambar 5 Contoh hyperlink dibuka melalui file explorer

BAB 2
LANDASAN TEORI

A. Algoritma Traversal Graf

Graf adalah kumpulan titik-titik/simpul yang saling dihubungkan oleh sebuah garis/sisi. Dalam dunia matematika, graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Sehingga, dapat dikatakan bahwa graf merupakan sebuah representasi dari suatu persoalan.

Dengan begitu, traversal dalam graf adalah pencarian solusi dari persoalan. Untuk melakukan traversal graf, terdapat algoritma traversal graf. Algoritma traversal graf berarti mengunjungi simpul-simpul pada graf dengan cara yang sistematik. Terdapat beberapa algoritma traversal graf, beberapa di antaranya yaitu pencarian melebar (breadth first search/BFS) dan pencarian mendalam (depth first search/DFS). Kedua algoritma tersebut digunakan dengan asumsi graf terhubung.

Algoritma traversal graf termasuk pada algoritma pencarian solusi. Berdasarkan informasi yang dimiliki, algoritma pencarian solusi dapat dibagi menjadi dua, yaitu tanpa informasi (uninformed/blind search) dan dengan informasi (informed search). Algoritma BFS dan DFS termasuk pada algoritma pencarian solusi tanpa informasi tambahan.

Selain itu, berdasarkan pendekatan pencarian solusi, graf dibagi menjadi dua, yaitu graf statis dan dinamis. Graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dilakukan. Pada umunya, graf statis direpresentasikan sebagai struktur data. Kedua, graf dinamis adalah graf yang terbentuk saat proses pencarian dilakukan. Oleh karena itu, graf belum terbentuk sebelum pencarian dan dibangung selama pencarian solusi.

B. Breadth First Search (BFS)

Breadth first search atau biasa disebut BFS adalah algoritma traversal graf yang melakukan pencarian pada graf dengan menelusuri simpul tiap level. Karena itu, BFS merupakan pencarian yang melebar. Secara umum, algortima BFS adalah sebagai berikut,

1. Mulai pencarian dari sebuah simpul, misal v.
2. Kunjungi simpul v.
3. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
4. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

Struktur data yang digunakan dalam algoritma BFS, yaitu

1. Matriks ketetanggaan (*adjacency matrix*) A = [a_{ij}] yang berukuran nxn, a_{ij} = 1, jika simpul i dan simpul j bertetangga, a_{ij} = 0, jika simpul i dan simpul j tidak bertetangga.

2. Antrian (queue) q untuk menyimpan simpul yang akan dikunjungi.
3. Tabel Boolean untuk mencatat simpul yang telah dikunjungi, misal nama tabel adalah “visited”. Maka
   a. visited: array[1..n] of boolean
   b. visited[i] = true, jika simpul i sudah dikunjungi
   c. visited[i] = false, jika simpul i belum dikunjungi

C. Depth First Search (DFS)

Depth first search atau biasa disebut DFS adalah algoritma traversal graf yang melakukan pencarian pada graf dengan menelusuri satu cabang sedalam mungkin hingga berhenti, kemudian melakukan *backtracking*. Algoritma dari DFS adalah sebagai berikut,

1. Misal pencarian dimulai dari simpul v.
2. Kunjungi tetangga simpul v, misal w.
3. Ulangi langkah 1 mulai dari simpul w.
4. Ketika pencarian sampai di sebuah simpul u sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtracking*) 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.

Untuk struktur data DFS hampir sama dengan BFS, hanya saja DFS tidak memerlukan sebuah antrian q, sehingga struktur data DFS menjadi,

1. Matriks ketetanggaan (*adjacency matrix*) A = [a_{ij}] yang berukuran nxn, a_{ij} = 1, jika simpul i dan simpul j bertetangga, a_{ij} = 0, jika simpul i dan simpul j tidak bertetangga.
2. Tabel Boolean untuk mencatat simpul yang telah dikunjungi, misal nama tabel adalah “visited”. Maka
   a. visited: array[1..n] of boolean
   b. visited[i] = true, jika simpul i sudah dikunjungi
   c. visited[i] = false, jika simpul i belum dikunjungi

D. C# Desktop Application Development

Desktop application development atau pengembangan aplikasi desktop adalah sebuah proses dimana pengembang membangun aplikasi yang dapat digunakan di laptop atau desktop. Ada banyak bahasa yang dapat digunakan untuk mengembangkan aplikasi desktop, salah satu bahasa yang paling banyak digunakan adalah bahasa C#. Bahasa C# banyak digunakan karena bahasa ini merupaka bahasa yang cukup advanced, type-safe, dan berorientasi objek. Bahasa ini digunakan dalam ekosistem .NET.

Dalam pengembangannya, terdapat beberapa *framework* atau kerangka aplikasi yang dapat digunakan. Beberapa di antaranya adalah WinForms dan Windows Presentation

Foundation atau WPF. WinForms adalah sebuah *class library* yang sudah ada dalam *framework* .NET semenjak awal dibuat. *Framework* WinForms adalah kerangka yang berorientasi event, sehingga semua elemen yang terlihat secara visual berasal dari kelas-kelas kontrol dan menunggu masukan pengguna untuk melanjutkan fungsi aplikasi.

Selain itu, *framework* WPF adalah kerangka yang disediakan dalam .NET *framework* dan digunakan untuk mengembangkan UI dari sebuah aplikasi desktop. Berbeda dengan WinForm, WPF berfokus dalam mengembangkan grafis dan UI dari aplikasi desktop. WPF mampu menyatukan komponen-komponen UI yang berbeda. Hal itulah yang menjadi komponen utama dari penggunaan WPF.

BAB 3
ANALISIS PEMECAHAN MASALAH

A. Langkah-langkah Pemecahan Masalah

A.1 Algoritma DFS

Algoritma traversal Depth First Search(DFS), kami menggunakan teknik rekursif sehingga tidak ada struktur data stack yang diimplementasikan secara eksplisit. Namun, struktur data stack diimplementasikan secara implisit dalam proses saat kode dieksekusi.

Dalam menyelesaikan masalah per level, kami melakukan hal sebagai berikut.

1. Membangkitkan seluruh file dan folder hanya pada level selanjutnya yang terdapat dalam simpul.
2. Untuk setiap file (pasti daun), dicocokkan apakah sama dengan nama file yang dicari atau tidak. Jika iya, tambahkan daun ke pohon ruang status. Jika tidak, file diabaikan.
3. Apabila hanya ingin dicari satu file saja (bukan *all occurrence*) dan file sudah ditemukan, maka rekursi akan berhenti.
4. Namun, apabila tidak ditemukan atau ingin mencari seluruh file, pencarian dilanjutkan. Untuk setiap nama folder yang sudah dibangkitkan, dilakukan pemanggilan rekursif untuk menelusuri folder selanjutnya secara DFS (ulang dari Langkah 1).
5. Secara singkat,
    a. Kasus basis:
        - Not all occurrence: ketika 1 file sudah ditemukan (diketahui dengan menyimpan nilai Boolean secara global).
        - All occurrence: ketika folder sudah tidak memiliki folder lain di dalamnya sehingga hanya akan dicek file saja.
    b. Kasus rekurens: ketika masih ada folder di dalam simpul folder yang dijalankan.

Perlu diperhatikan juga karena tiap anak dari simpul dijalankan secara iteratif, maka fungsi rekursif yang masuk hanya anak pertama saja, sedangkan sisanya masih belum dipanggil DFS-nya sehingga **tidak ada di dalam stack**. Dengan kata lain, dapat dikatakan bahwa DFS “mengunjungi anak” terlebih dahulu, baru “membangkitkan anak berikutnya” sehingga tidak ada simpul yang hitam pada persoalan DFS.

A.2 Algoritma BFS

Pada algoritma Breadth First Search(BFS), kami menggunakan struktur data queue untuk menyimpan folder dan file yang akan dicek. Maka untuk setiap iterasinya, kami melakukan hal sebagai berikut :

1. Membangkitkan daftar file dan subdirektori pada folder yang sedang diproses.
2. Untuk setiap file, dicocokkan apakah sama dengan nama file yang dicari atau tidak. Jika sama, tambahkan daun ke pohon ruang status, dan beri warna hijau untuk semua directory parent dari startingDir hingga direktori sekarang. Jika tidak sama, maka tambahkan daun yang menandakan file tidak cocok.
3. Apabila hanya ingin dicari satu file saja, semua folder dan file lain yang belum dicek (antrian dalam queue) akan dibuat daun, namun diberi warna hitam yang berarti folder dan file belum dicek. Iterasi akan langsung dihentikan dengan mengubah nilai variabel fileFound menjadi true.
4. Namun, apabila belum ditemukan atau ingin mencari seluruh file, pencarian file dilanjutkan dan folder yang sebelumnya dibangkitkan jika belum ada di dalam queue akan dimasukkan ke dalam queue dan akan ditambahkan daun pada pohon status dengan warna hitam untuk menandakan belum dicek.

B. Proses Mapping Persoalan ke Dalam Elemen BFS dan DFS

Pada persoalan penelusuran file dan direktori, pendekatan pencarian solusi dilakukan dengan graf dinamis, yaitu graf yang terbentuk saat pencarian dilakukan. Graf atau pohon dinamis direpresentasikan dengan pohon ruang status, simpul, cabang, ruang status, dan ruang solusi.

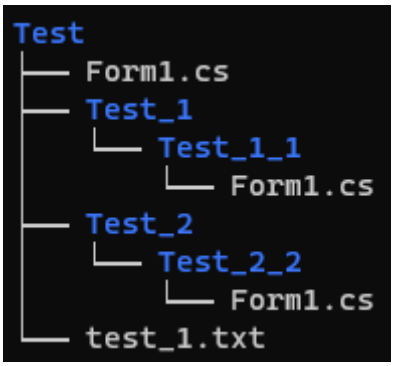
Simpul akar pada pohon melambangkan *initial state* dan simpul daun pada pohon melambangkan *solution/goal state*. Cabang dalam pohon melambangkan operator dari persoalan atau langkah yang diambil dalam pencarian. Ruang solusi adalah himpunan status solusi dari persoalan. Ruang status adalah himpunan dari semua simpul pada pohon sehingga pohon ruang status adalah sebutan untuk pohon yang semua simpulnya merupakan anggota ruang status.

Berdasarkan definisinya, pemetaan elemen-elemen BFS/DFS adalah sebagai berikut.

1. Cabang/operator: tambah folder/file
2. Akar: direktori/folder awal yang dipilih
3. Simpul: direktori/folder yang berada di dalam folder akar
4. Daun: file yang dicari atau direktori yang sudah tidak memiliki folder di dalamnya
5. Ruang solusi: himpunan semua direktori file yang ditemukan
6. Ruang status: seluruh simpul di dalam pohon dan pohonnya dinamakan juga pohon ruang status

C. Contoh Ilustrasi Kasus

Contoh kasus lain yang kami buat sendiri selain dari spesifikasi adalah pencarian Form1.cs dalam folder root Test. Dalam folder root Test ini, terdapat 3 file bernama Form1.cs.



Gambar 6 Struktur folder Test

Untuk mencari 1 file saja, dengan menggunakan BFS dan DFS maka akan langsung ditemukan Form1.cs yang ada di root folder test.

Berikut diagram pencarian satu file menggunakan algoritma BFS

Gambar 7 Ilustrasi hasil pencarian menggunakan algoritma BFS

Berikut diagram pencarian satu file menggunakan algoritm DFS

Gambar 8 Ilustrasi hasil pencarian menggunakan algoritma DFS

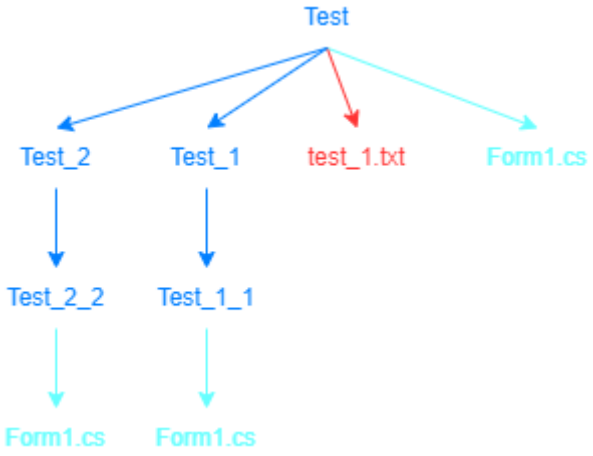
Kedua algoritma tersebut menghasilkan *file path* yang sama, yaitu

1. `\Test`

Sedangkan untuk pencarian semua file bernama Form1.cs, dengan menggunakan BFS pertama akan ditambahkan kedalam queue semua directory yang ada di root test (Test_1, Test_2, Form1.cs, test_1.txt) dan akan dicocokan dengan target filenya yaitu Form1.cs, lalu kemudian akan mengecek ke dalam directory selanjutnya, dalam hal ini Test_1 dan akan dilakukan hal yang sama hingga queue pengecekan sudah habis.

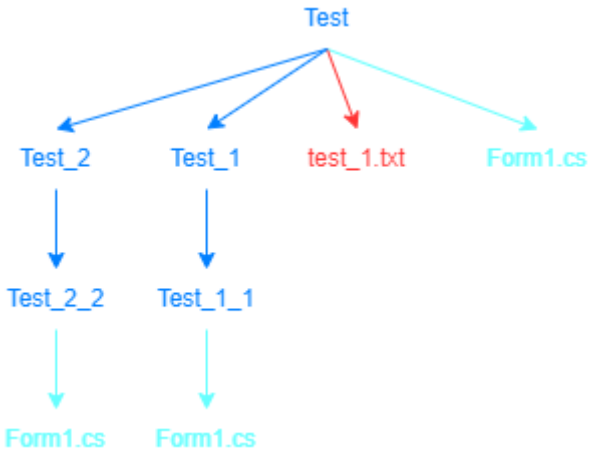
Sedangkan dengan DFS, akan digunakan rekursif untuk terus mencari di directory pertama dalam root test hingga tidak ditemukan lagi directory lain, baru berpindah ke directory selanjutnya dalam root test.

Berikut diagram pencarian semua file menggunakan algoritma BFS



Gambar 9 Ilustrasi hasil pencarian menggunakan algoritma BFS

Berikut diagram pencarian semua file menggunakan algoritma DFS



Gambar 10 Ilustrasi hasil pencarian menggunakan algoritma BFS

Kedua algoritma tersebut menghasilkan file path yang sama, yaitu

1. `\Test`
2. `\Test\Test_1\Test_1_1`
3. `\Test\Test_2\Test_2_2`

BAB 4
IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Program

A.1 Pseudocode Algoritma DFS

Perhatikan bahwa currentNode adalah path file yang dijalankan. Kemudian, terdapat Boolean found yang didefinisikan global. Agar algoritmanya lebih sederhana, pergantian warna simpul dan sisi tidak dimasukkan.

```
function dfs (string currentNode, string searchedFile, bool
isAllOccurrence) → bool (ditemukan / tidak)
    isInFileChild ← false; isInFolderChild ← false
    dirList ← ambil top directory dari currentNode
    fileList ← ambil all files dari currentNode

    for setiap file yang ada di dalam fileList:
      if nama file = searchedFile then
            tambah edge dari currentNode ke file
            found ← true
            if found and not(isAllOccurrence) then
                  → true

      else { do nothing }


    if not(found) or isAllOccurrence then
          for setiap top yang ada di dalam dirList:
              tambah edge dari currentNode ke top
              temp ← dfs(top, searchedFile, isAllOccurrence)
              if (temp) then
                  isInFolder ← true

    → (isInFileChile || isInFolderChild)
```

A.2 Pseuducode Algoritma BFS

Agar algoritma lebih sederhana, pergantian warna simpul dan sisi tidak dimasukkan.

```
procedure bfs (string startingDir, string filename, bool
isAllOccurrence)
    buat queue findQue
    buat array doneCheck {menyimpan daftar folder sudah dicek}

    push startingDir into findQue
    bool fileFound ← false

    while queue tidak kosong and fileFound=false
        checking ← keluarkan elemen dari findQue
        dirList ← ambil all directories dari currentNode
```

```
fileList ← ambil all files dari currentNode

for setiap file yang ada di dalam fileList:
    if nama file = searchedFile then
        tambah edge dari currentNode ke file
        found ← true
    else { do nothing }

push checking into doneCheck

for setiap folder yang ada di dalam dirList:
    push folder into findQue
```

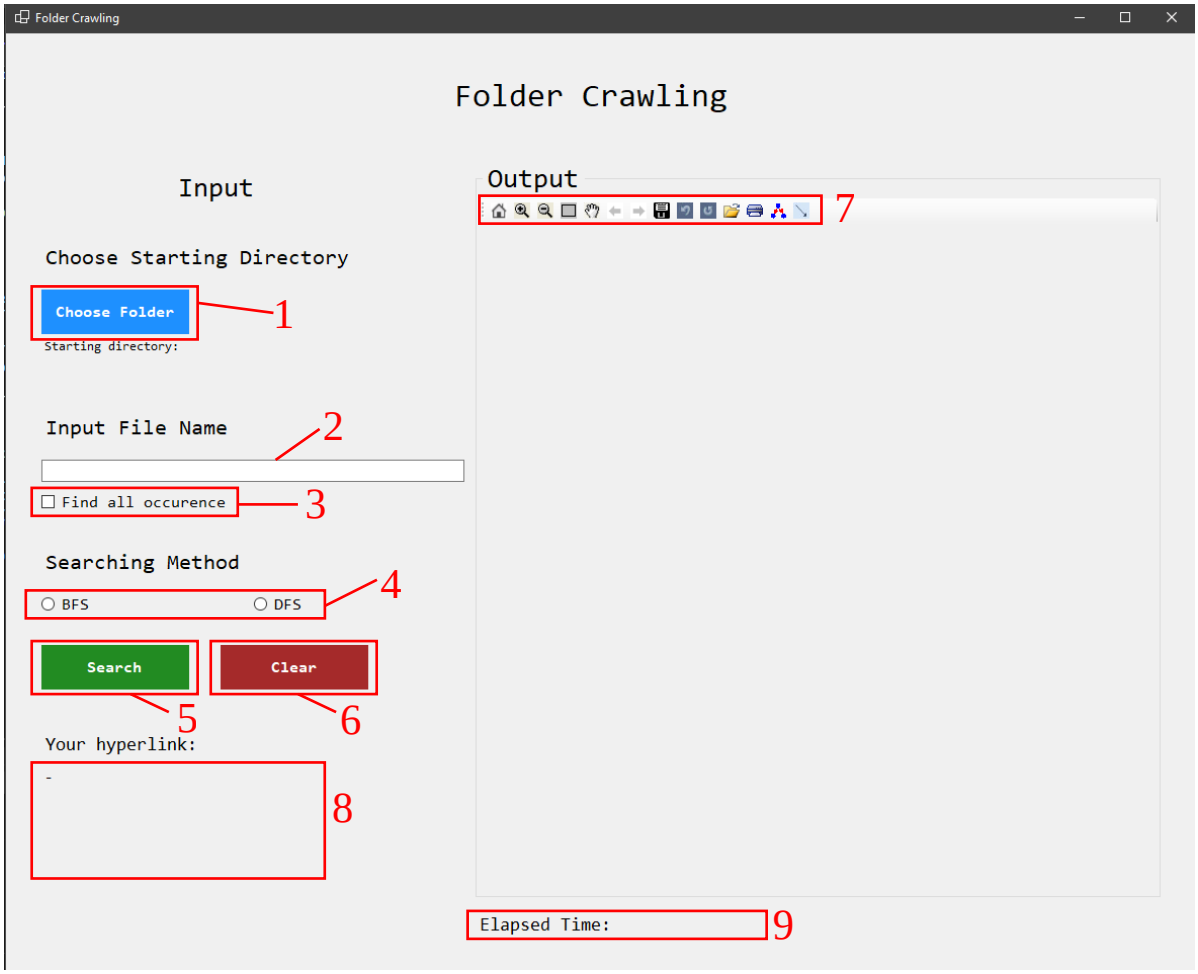
B. Struktur Data dalam Program dan Spesifikasi Program

Struktur data yang digunakan dalam program adalah:

- List
  Struktur data list dalam program digunakan untuk menyimpan tipe data string. Metode yang dapat dilakukan oleh struktur data ini adalah menambah elemen ke dalam List, menghapus elemen, mengubah ukuran List, dan mengakses elemen List.
- Stack
  Stack diimplementasikan secara implisit saat mengeksekusi algoritma DFS dengan menggunakan "call stack" dari fungsi rekursif yang dipanggil.
- Queue
  Queue dalam program digunankan untuk menyimpan string direktori atau folder yang akan dicari dalam algoritma BFS. Metode yang dapat dilakukan oleh struktur data Queue adalah menambahkan elemen ke dalam antrian, menghilangkan elemen pertama dari antrian, serta mengakses elemen pertama antrian.

Spesifikasi yang dimiliki program yaitu program mampu memilih direktori/folder awal yang akan menjadi akar atau *initial state* dari graf. Kemudian, program mampu menerima masukan dari pengguna berupa nama *file* yang ingin dicari. Program dapat memberikan pilihan algoritma yang dapat dipakai untuk mencari solusi, yaitu BFS atau DFS. Program dapat menerima permintaan mencari semua solusi atau semua *file* yang dicari. Setelah itu, program akan menampilkan pohon atau graf dari runutan pencarian solusi serta *hyperlink* dari *file* jika ditemukan.

C. Tata Cara Penggunaan Program



Gambar 11 Interface program

Gambar di atas merupakan interface dari program, berikut adalah penjelasan dari masing masing komponen dan fitur yang telah diberikan nomor:

1. Tombol Choose Folder, tombol tersebut berfungsi untuk memilih direktori awal tempat pencarian.
2. *Field* Input File Name, pengguna memberikan masukan nama file yang akan dicari ke dalam *field* tersebut.
3. *Checkbox* Find All Occurrence (opsional), jika dicentang, maka program akan mencari semua kemunculan file di semua folder di bawah direktori awal.
4. Tombol radio BFS dan DFS, pengguna harus memilih salah satu dari keduanya. Jika pengguna memilih BFS, maka program akan mencari file dengan algoritma BFS. Sebaliknya, program akan mencari file menggunakan algoritma DFS.
5. Tombol Search, jika tombol ditekan ketika masukan yang diperlukan telah terpenuhi, program akan mulai mencari file sesuai dengan masukan pengguna.
6. Tombol Clear, tombol ini akan menghapus *hyperlink* yang telah dibuat (jika ada), me-*reset* waktu yang ditampilkan pada label Elapsed Time, dan menghapus graf yang sudah dibuat oleh program.

7. Tombol-tombol tersebut digunakan untuk melakukan manipulasi terhadap graf yang dihasilkan. Urut dari kiri ke kanan, tombol-tombol tersebut adalah Home, Zoom In, Zoom Out, Zoom In by Dragging a Rectangle, Pan, Backward, Forward, Save, Undo, Redo, Load, Print, Graph Configuration, dan AddEdge. Tombol Zoom In digunakan untuk memperbesar graf. Tombol Zoom Out digunakan untuk memperkecil ukuran graf. Tombol Zoom In by Dragging a Rectangle digunakan untuk memperbesar graf pada segi empat yang dibuat pada graf. Tombol Pan digunakan untuk menggeser graf secara keseluruhan. Tombol Backward digunakan untuk menggeser gambar ke belakang. Tombol Forward digunakan untuk menggeser gambar ke depan. Tombol Save digunakan untuk menyimpan graf yang telah dibuat menjadi sebuah file image. Tombol Undo digunakan untuk mengembalikan graf ke kondisi sebelum diedit. Tombol Redo digunakan untuk melakukan ulang manipulasi yang dilakukan terhadap graf. Tombol Load digunakan untuk memuat file graf lain. Tombol Print digunakan untuk melakukan print graf yang sudah dibuat. Tombol Graph Configuration digunakan untuk mengatur pengaturan graf. Tombol AddEdge digunakan untuk menambahkan sisi pada graf.
8. *Field* Hyperlink, *hyperlink* kepada file yang ditemukan akan ditampilkan pada *field* tersebut.
9. Label Elapsed Time, label tersebut akan menampilkan waktu yang dibutuhkan program untuk mencari file.

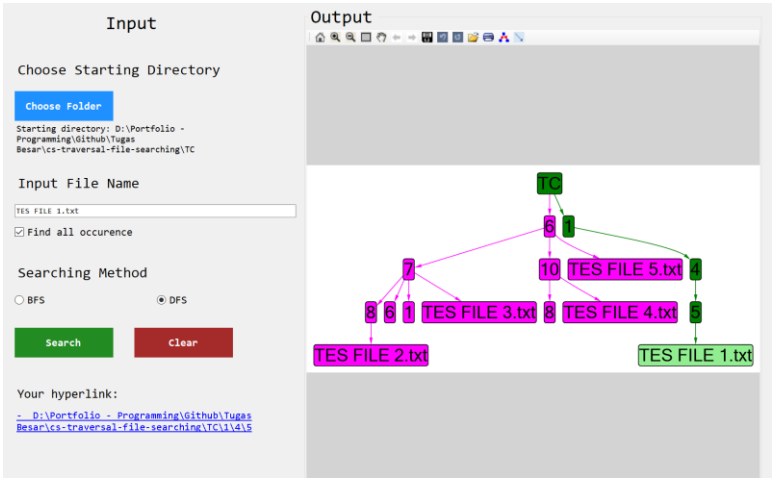
Langkah-langkah penggunaan program adalah sebagai berikut:

1. Pilih direktori awal pencarian dengan menekan tombol Choose Folder. Sebuah window pemilihan folder akan terbuka, jika sudah memilih, tekan tombol OK.
2. Masukan nama file yang ingin dicari pada *field* Input File Name.
3. Jika ingin mencari semua kemunculan file, centang Find all occurrence.
4. Pilih salah satu algoritma pencarian, yaitu BFS atau DFS.
5. Tekan tombol Search.
6. Ketika program sudah selesai mencari, akan muncul waktu yang dihabiskan oleh program pada label Elapsed Time. Graf yang dibuat akan muncul pada *field* Output. Hyperlink menuju direktori file akan muncul pada *field* Your Hyperlink.
7. Tekan tombol Clear untuk menghapus hyperlink, elapsed time, dan graf yang sudah dibuat.

D. Hasil Pengujian

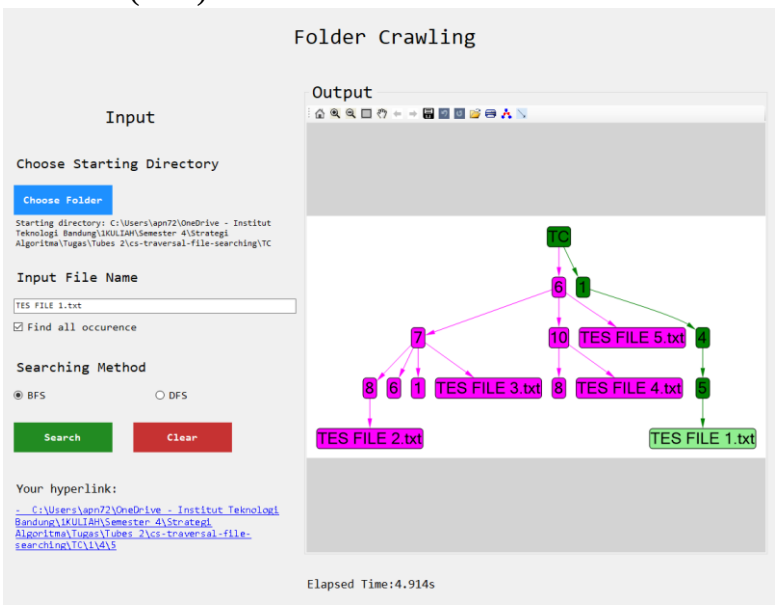
1. Kasus 1: ada nama folder yang sama, tetapi file tidak.

a. All occurrence (DFS)



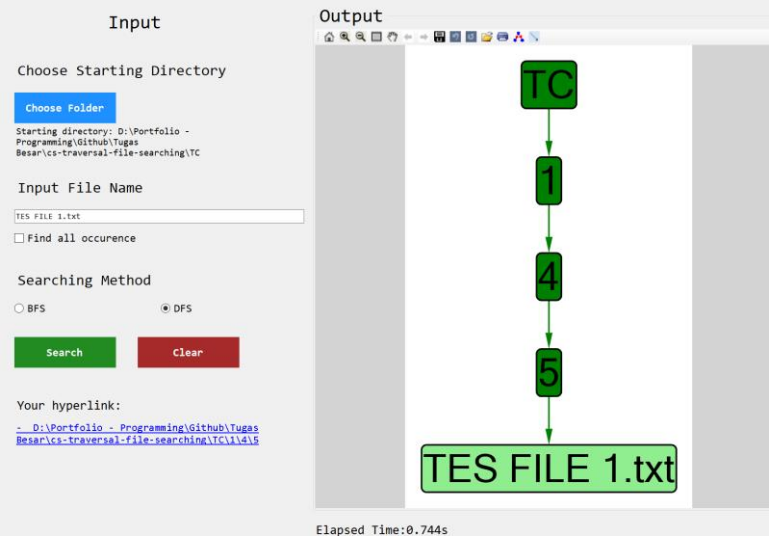
Gambar 12 Kasus nama folder sama untuk algoritma DFS all occurence

b. All occurrence (BFS)



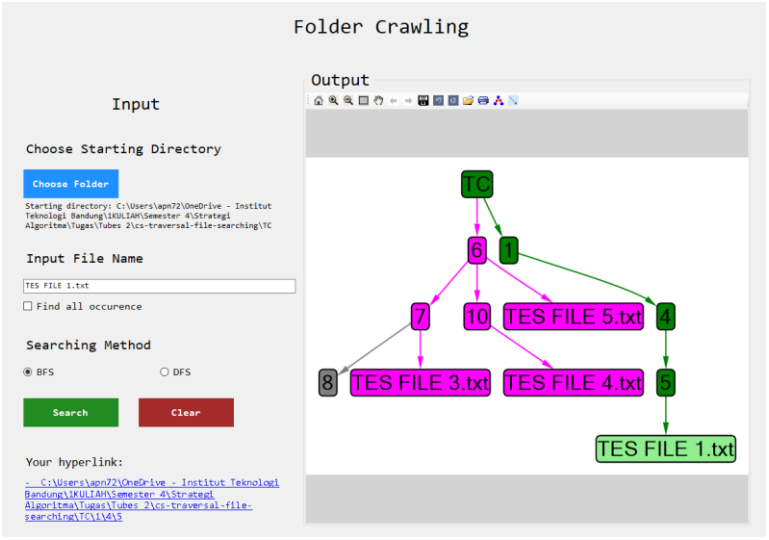
Gambar 13 Kasus nama folder sama untuk algoritma BFS all occurence

c. One occurrence (DFS)



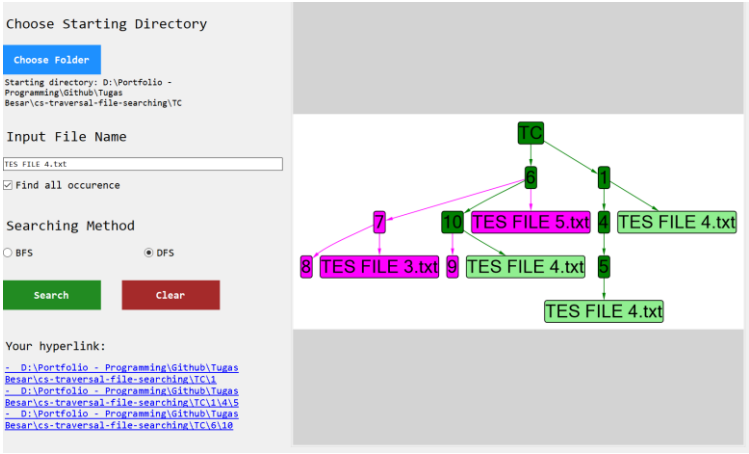
Gambar 14 Kasus nama folder sama untuk algoritma DFS one occurence

d. One occurrence (BFS)



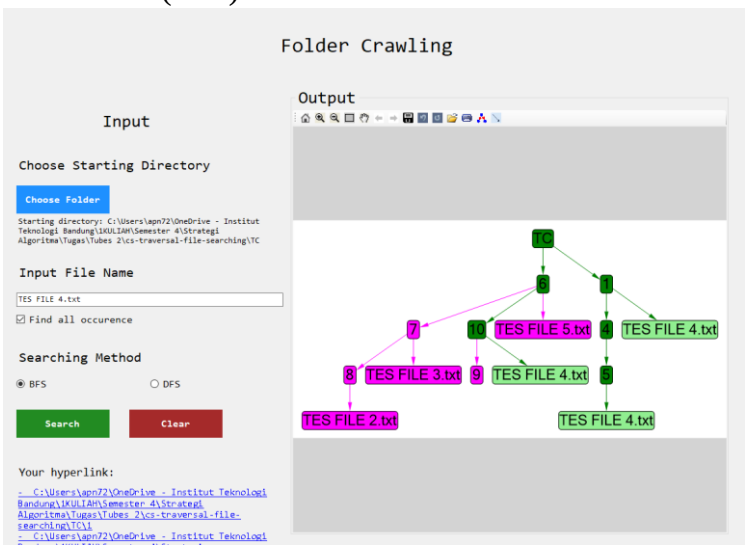
Gambar 15 Kasus nama folder sama untuk algoritma BFS one occurence

2. Kasus 2: ada nama file yang sama, tetapi folder tidak.
   a. All occurrence (DFS)



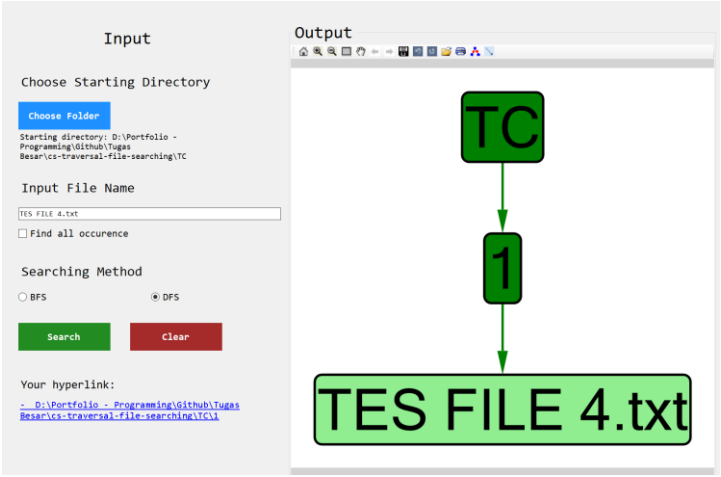
Gambar 16 Kasus nama file sama untuk algoritma DFS all occurence

   b. All occurrence (BFS)



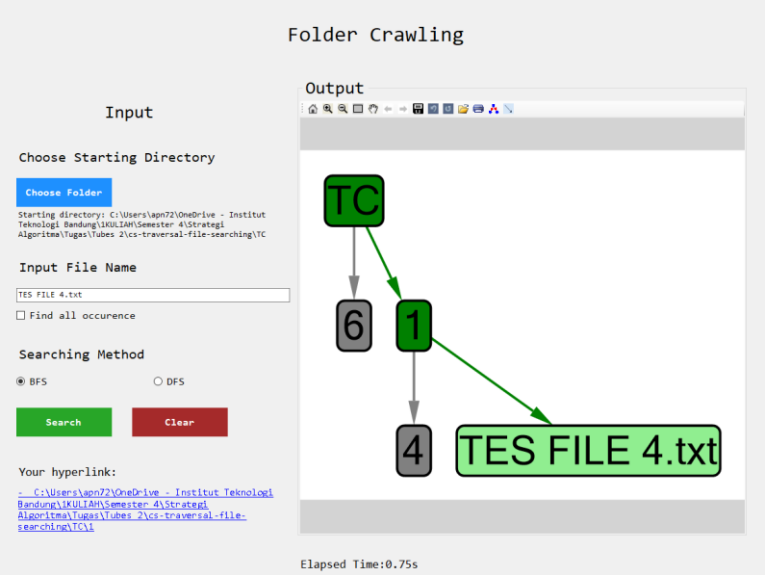
Gambar 17 Kasus nama file sama untuk algoritma BFS all occurence

   c. One occurrence (DFS)



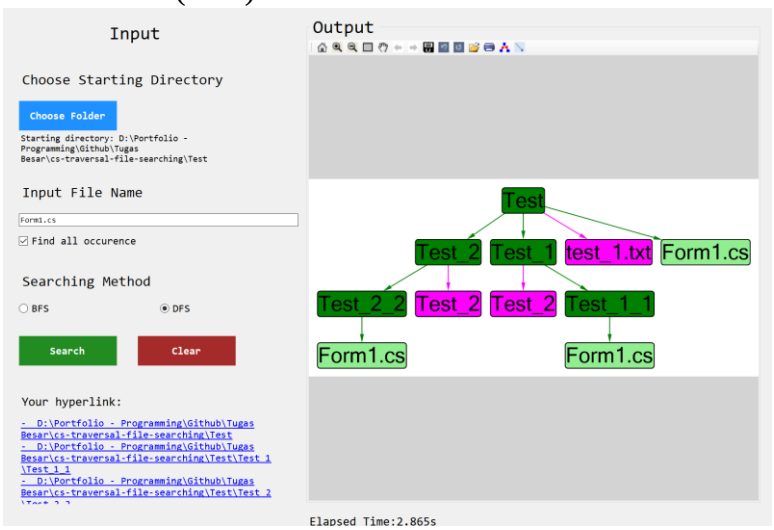
Gambar 18 Kasus nama file sama untuk algoritma DFS one occurence

d. One occurrence (BFS)



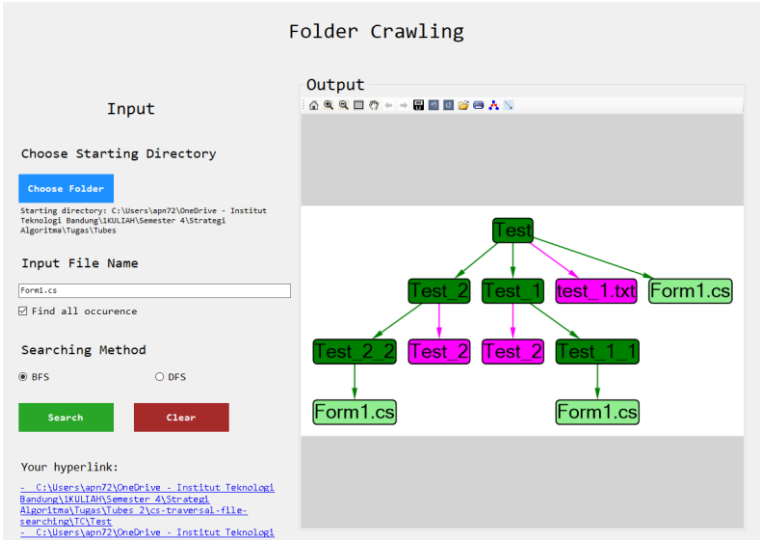
Gambar 19 Kasus nama file sama untuk algoritma BFS one occurence

3. Kasus 3: ada nama folder dan file yang sama.
   a. All occurrence (DFS)



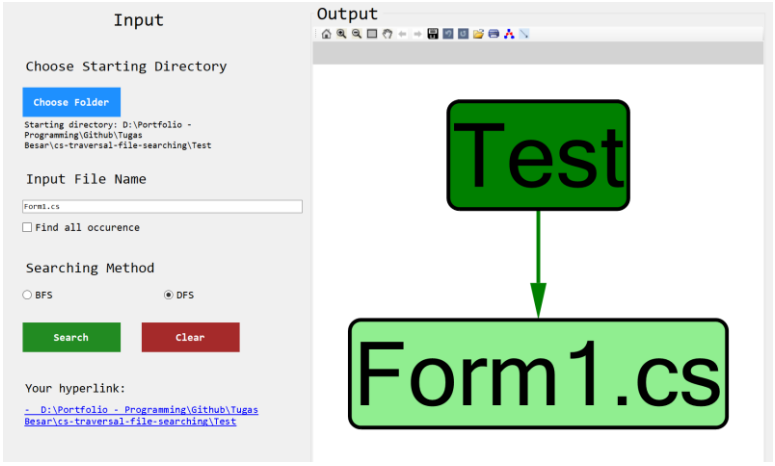
Gambar 20 Kasus nama folder dan file sama untuk algoritma DFS all occurence

b. All occurrence (BFS)



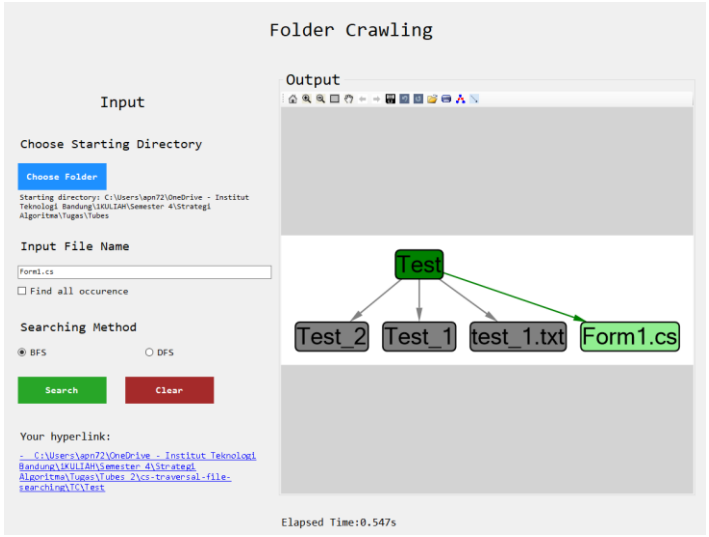
Gambar 21 Kasus nama folder dan file sama untuk algoritma BFS all occurence

c. One occurrence (DFS)



Gambar 22 Kasus nama folder dan file sama untuk algoritma DFS one occurence

d. One occurrence (BFS)



Gambar 23 Kasus nama folder dan file sama untuk algoritma BFS one occurence

E. Analisis Desain Solusi Algoritma BFS dan DFS

Dilihat dari notasi O, algoritma BFS memiliki kompleksitas waktu yang lebih baik dari DFS, yaitu O(b^d) untuk BFS dan O(b^m) untuk DFS dengan b adalah jumlah maksimum anak, d adalah kedalaman dari solusi, dan m adalah maksimal kedalaman dari pohon. Namun, kompleksitas ruang BFS lebih besar daripada DFS karena dengan algoritma BFS karena struktur data antrian berkembang (grow) lebih cepat daripada stack milik DFS. Kompleksitas ruang BFS adalah O(b^d) dan DFS adalah O(bm).

Sebagai contoh, dilakukan pencarian terhadap file "TES FILE 1.txt" dan didapatkan graf pada gambar 11 dan 12. Pencarian file tersebut menggunakan algoritma DFS akan membentuk urutan, yaitu TC → 1 → 4 → 5 → "TES FILE 1.txt". Sedangkan pencarian menggunakan algoritma BFS akan membentuk urutan penelusuran, yaitu TC → 1 → 6 → "TES FILE 5.txt" → 7 → "TES FILE 3.txt" → 10 → "TES FILE 4.txt" → 5 → "TES FILE 1.txt". Hasil urutan penelusuran tersebut membuktikan bahwa kompleksitas ruang pada BFS lebih besar dibandingkan dengan DFS. Akan tetapi, hasil urutan penelusuran tersebut juga menunjukkan bahwa kompleksitas waktu pada algoritma DFS lebih baik dibandingkan dengan algoritma BFS untuk kasus tersebut, yaitu kasus dimana file yang dicari berada di folder pertama dari anak.

BAB 5
KESIMPULAN DAN SARAN

A. Kesimpulan

Algoritma DFS dan BFS merupakan contoh algoritma untuk melakukan implementasi folder crawling. Kedua algoritma tersebut menggunakan pendekatan yang berbeda untuk melakukan folder crawling, dimana BFS akan mengecek yang berada di sekitar terlebih dahulu sedangkan DFS akan mengecek yang lebih dalam terlebih dahulu. Karena pendekatannya berbeda, kedua algoritma memiliki kelebihan dan kekurangan pada beberapa kasus spesial tertentu. Meskipun begitu, secara umum tidak ada algoritma yang lebih baik dari yang lainnya dalam konteks folder crawling. Selain itu, kedua algoritma akan menghasilkan hasil yang sama pada semua kasus.

B. Saran

Untuk menyelesaikan masalah serupa, dapat dicoba menggunakan algoritma lain seperti IDS (Iterative Deepening Search) atau DLS (Depth Limited Search). Meski demikian, perlu diperhatikan juga karakteristiknya. Kedua algoritma tersebut bisa saja lebih baik dari segi kompleksitas waktu dan ruang, tetapi bisa saja tidak menjamin ditemukannya solusi (tidak komplit). Selain itu, karena struktur folder berupa tree, bisa saja digunakan algoritma traversal tree. Jadi, tidak umum graf seperti BFS atau DFS.

DAFTAR PUSTAKA

Kemdikbud. 2022. *Pengertian Graf*. Diakses pada 16 Maret 2022, dari https://lmsspada.kemdikbud.go.id/mod/resource/view.php?id=47638

Munir, Rinaldi & Maulidevi, Nur U. 2022. *Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1)*. Diakses pada 16 Maret 2022, dari http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf

Munir, Rinaldi & Maulidevi, Nur U. 2022. *Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 2)*. Diakses pada 16 Maret 2022, dari http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf