Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder Crawling*

LAPORAN TUGAS BESAR

Diajukan Untuk Memenuhi Tugas Besar 2 IF2211 Strategi Algoritma Semester II Tahun 2021/2022

Disusun oleh:

Kelompok FNF

13520038	**Shadiq Harwiz**
13520125	**Ikmal Alfaozi**
13520138	**Gerald Abraham Sianturi**

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

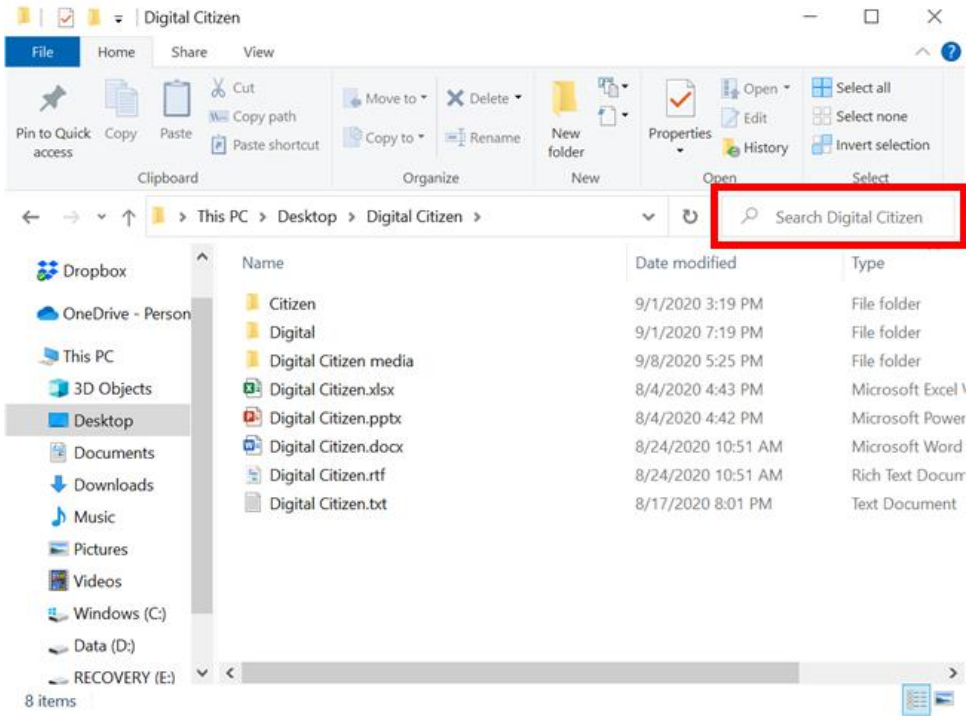
BANDUNG

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.



Gambar 1. Fitur Search pada Windows 10 File Explorer
(Sumber: https://www.digitalcitizen.life/wp-content/uploads/2020/10/explorer_search_10.png)

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.

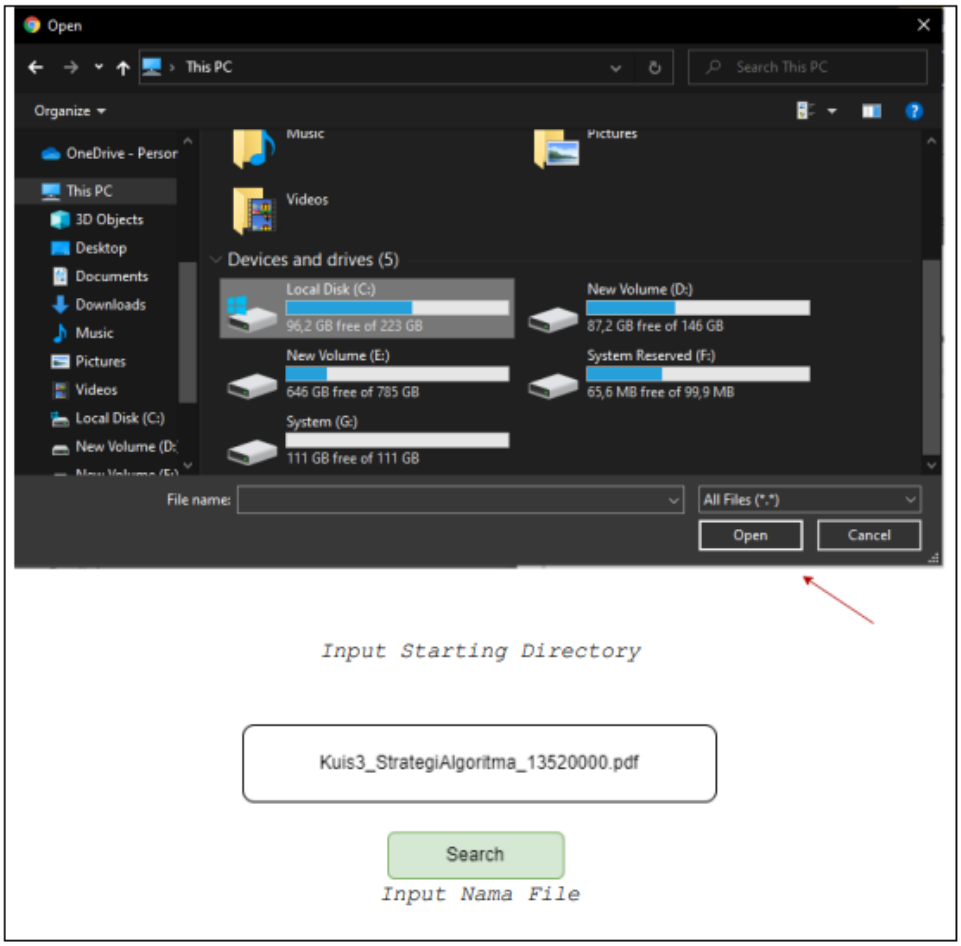
Fitur ini diimplementasikan dengan teknik folder crawling, di mana mesin komputer akan mulai mencari file yang sesuai dengan query mulai dari starting directory hingga seluruh children dari starting directory tersebut sampai satu file pertama/seluruh file ditemukan atau tidak ada file yang ditemukan. 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.

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.

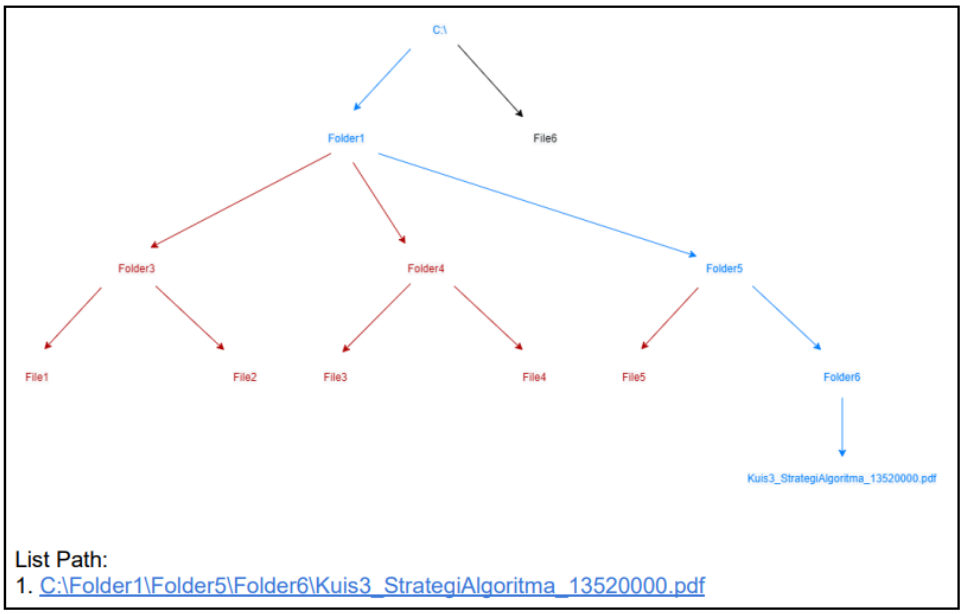
Selain pohon, Anda diminta juga menampilkan list path dari daun-daun yang bersesuaian dengan hasil pencarian. Path tersebut diharuskan memiliki hyperlink menuju folder parent dari file yang dicari, agar file langsung dapat diakses melalui browser atau file explorer. Contoh hal-hal yang dimaksud akan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

Contoh Input dan Output Program Contoh masukan aplikasi:



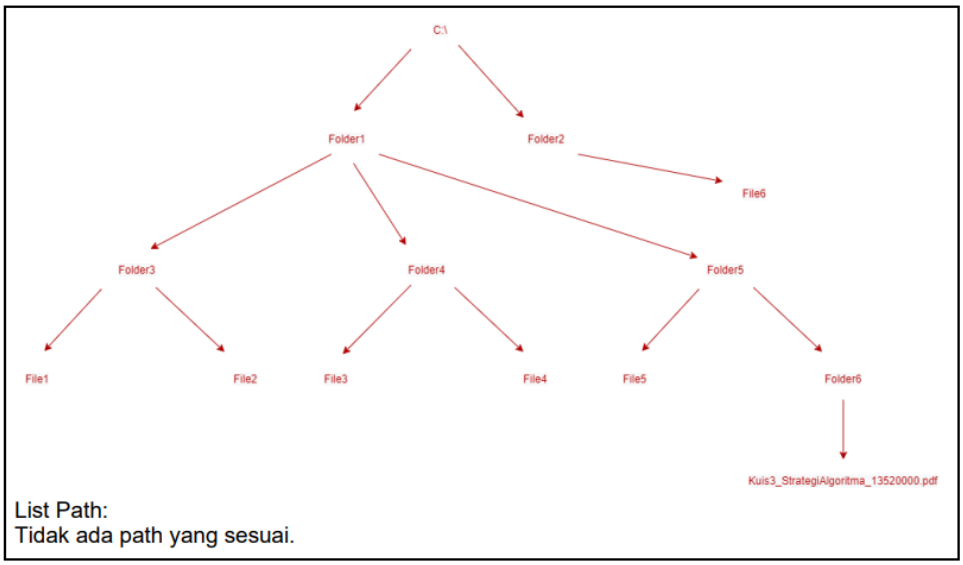
Gambar 2. Contoh input program

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.

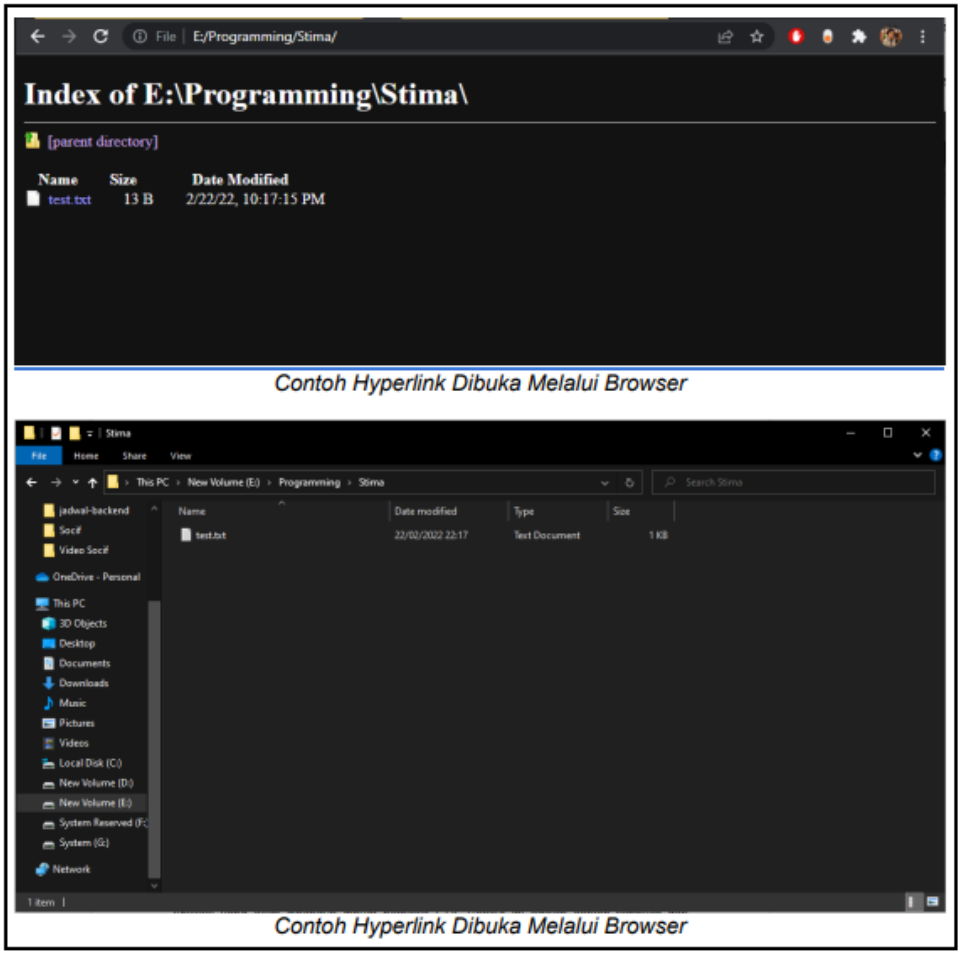


Gambar 4. Contoh output program jika file tidak ditemukan

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.

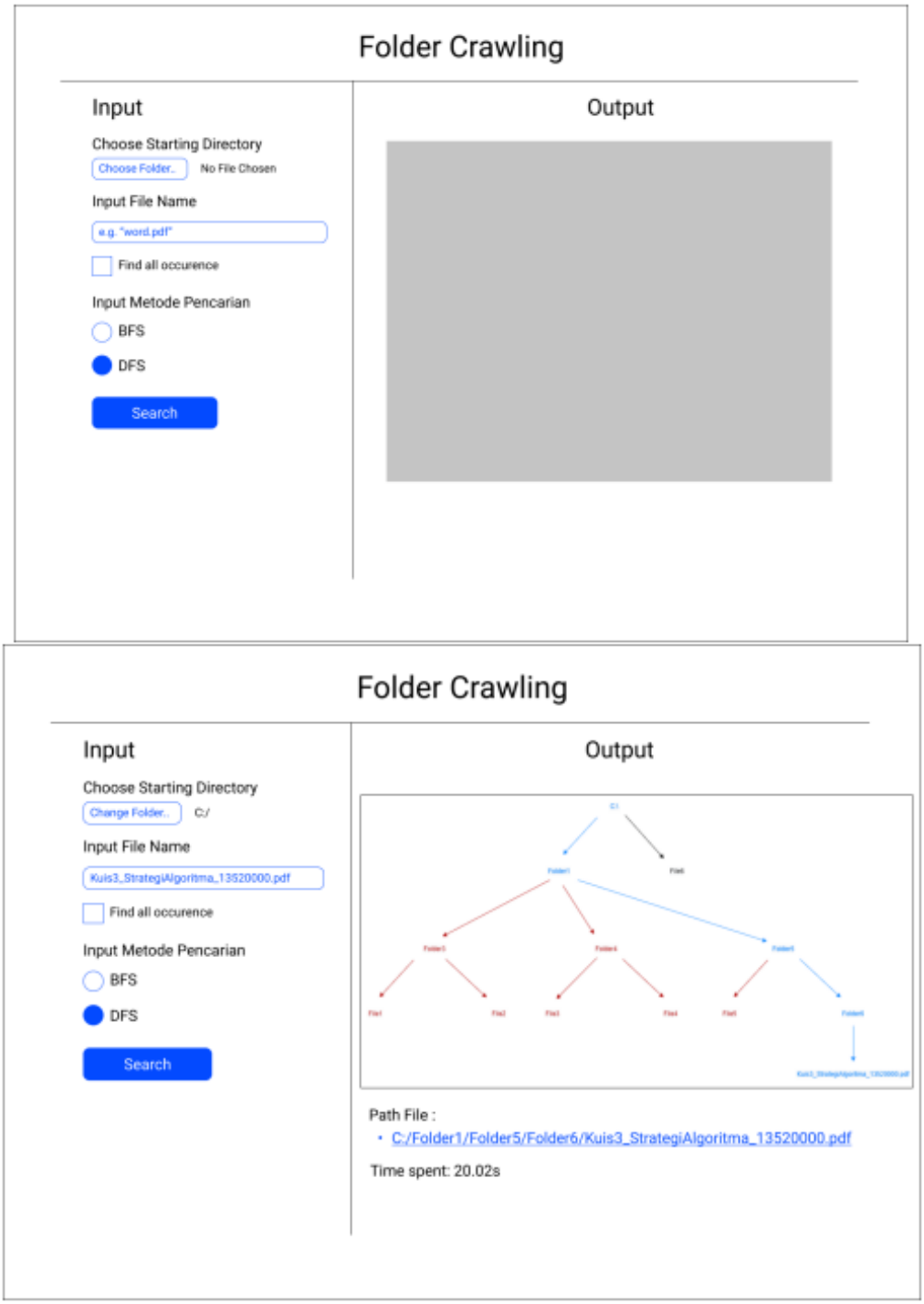
Contoh Hyperlink Pada Path:



Gambar 4. Contoh ketika hyperlink di-klik

Spesifikasi Program:

Aplikasi yang akan dibangun dibuat berbasis GUI. Berikut ini adalah contoh tampilan dari aplikasi GUI yang akan dibangun.



Gambar 9. Tampilan layout dari aplikasi desktop yang dibangun

Catatan: Tampilan diatas hanya berupa salah satu contoh layout dari aplikasi saja, untuk design layout aplikasi dibebaskan dengan syarat mengandung seluruh input dan output yang terdapat pada spesifikasi.

Spesifikasi GUI:

1. Program dapat menerima input folder dan query nama file.
2. Program dapat memilih untuk menampilkan satu hasil saja atau menemukan semua file yang memiliki nama file sama persis dengan input query
3. Program dapat memilih algoritma yang digunakan.

4. Program dapat 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 serta file yang merupakan rute hasil pertemuan.
5. (Bonus) Program dapat menampilkan progress pembentukan pohon dengan menambahkan node/simpul sesuai dengan pemeriksaan folder/file yang sedang berlangsung.
6. Program dapat menampilkan hasil pencarian berupa rute/path (bisa lebih dari satu jika memilih menemukan semua file) serta durasi waktu algoritma.
7. GUI dapat dibuat **sekreatif** mungkin asalkan memuat 5(6 jika mengerjakan bonus) spesifikasi di atas.

Program yang dibuat harus memenuhi **spesifikasi wajib** sebagai berikut:

1) Buatlah program dalam bahasa C# untuk melakukan penelusuran Folder Crawling sehingga diperoleh hasil pencarian file yang diinginkan. Penelusuran harus memanfaatkan algoritma BFS dan DFS.
2) Awalnya program menerima sebuah input folder pada direktori yang ada dan nama file yang akan dicari oleh program.
3) Terdapat dua pilihan pencarian, yaitu:
    a. Mencari 1 file saja Program akan memberhentikan pencarian ketika sudah menemukan file yang memiliki nama sama persis dengan input nama file.
    b. Mencari semua kemunculan file pada folder root Program akan berhenti ketika sudah memeriksa semua file yang terdapat pada folder root dan program akan menampilkan daftar semua rute file yang memiliki nama sama persis dengan input nama file.
4) Program kemudian dapat menampilkan visualisasi pohon pencarian file berdasarkan informasi direktori dari folder yang di-input. Pohon hasil pencarian file ini memiliki root adalah folder yang di-input dan setiap daunnya adalah file yang ada di folder root tersebut. Setiap folder/file direpresentasikan sebagai sebuah node atau simpul pada pohon. Cabang pada pohon menggambarkan folder/file yang terdapat di folder parent-nya.

    Visualisasi pohon juga harus disertai dengan keterangan node yang sudah diperiksa, node yang sudah masuk antrian tapi belum diperiksa, dan node yang bagian dari rute hasil penemuan.

    Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk melakukan visualisasi adalah MSAGL (https://github.com/microsoft/automatic-graph-layout) Berikut ini adalah panduan singkat terkait penggunaan MSAGL oleh tim asisten yang dapat diakses pada: https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVl3MY6gt1tPL30LA/edit?usp=sharing.

5) Program juga dapat menyediakan hyperlink pada setiap hasil rute yang ditemukan. Hyperlink ini akan membuka folder parent dari file yang ditemukan. Folder hasil hyperlink dapat dibuka dengan browser atau file explorer.
6) Mahasiswa tidak diperkenankan untuk melihat atau menyalin library lain yang mungkin tersedia bebas terkait dengan pemanfaatan BFS dan DFS. Tapi untuk algoritma lainnya seperti string matching dan akses directory, diperbolehkan menggunakan library jika ada.

Bab 2
Landasan Teori

2.1. Dasar Teori

2.1.1.1. Algoritma Traversal Graf

Traversal graf adalah proses mengunjungi (memeriksa/mengubah) setiap node di dalam graf. Persoalan pada algoritma traversal graf direpresentasikan dalam bentuk graf (diasumsikan graf terhubung). Terdapat dua algoritma yang paling sering digunakan untuk traversal graf, yaitu:

a. Pencarian melebar (breadth-first search/BFS)
b. Pencarian mendalam (depth-first search/DFS)

Dalam proses pencarian graf terdapat dua pendekatan, yaitu:

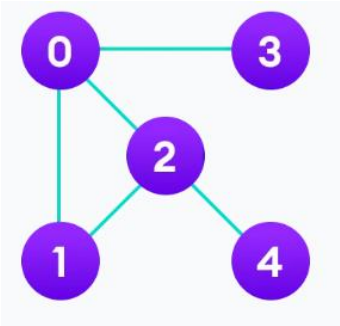
a. Graf statis
   Graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dan direpresentasikan sebagai struktur data.

b. Graf dinamis
   Graf dinamis adalah graf yang terbentuk saat proses pencarian. Graf tidak tersedia sebelum proses pencarian dilakukan dan dibangun ketika proses pencarian berlangsung.

2.1.2. *Breadth First Search* (BFS)

Breadth-first search (BFS) adalah algoritma pencarian sebuah node yang memenuhi kondisi tertentu pada struktur data *tree*. Pencarian umumnya dimulai dari *node* akar (*root*), kemudian dilakukan pencarian ke *node-node* pada level selanjutnya.
Proses traversal yang dilakukan pada algoitma ini, yakni:

a) Pertama, kunjungi simpul awal, misal simpul r,
b) Lalu, kunjungi semua simpul level selanjutnya dari simpul r, yakni simpul-simpul yang bertetangga dengan simpul *r,* Prioritas pemilihan simpul yang terlebih dahulu dikunjungi pada satu level yang sama dapat dilakukan sesuai preferensi, atau umumnya dilakukan penomoran prioritas,
c) Dari simpul-simpul tersebut, lakukan kembali poin *b* dimulai dari simpul yang terlebih dahulu dikunjungin pada level sebelumnya.



Gambar 10 Visualisasi Graf
Sumber: https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

Proses traversal pada instansiasi graf pada gambar 10:

a) Tentukan simpul awal dilakukannya traversal, yakni misalnya simpul yang ditandai dengan 0.
b) Lalu, kunjungi simpul pada level selanjutnya, yakni simpul yang bertetangga dengan simpul awal. Misal prioritas telah ditentukan sedemikian rupa sehingga simpul yang dikunjungi pada level ini berturut-turut adalah simpul yang ditandai dengan 1, 2, dan 3.

c) Karena simpul yang sebelumnya dikunjungi adalah simpul 1, kunjungi simpul yang bertetangga dengan simpul 1 (dalam kasus ini tidak ada)
d) Selanjutnya, cek simpul tetangga dari simpul 2, dan kunjungi semua simpul tersebut. Dalam kasus ini hanya terdapat satu simpul tetangga, yakni simpul 4
e) Kunjungi simpul tetangga dari simpul 3 (dalam kasus ini hanya simpul 0 yang ternyata sudah dikunjungi)
f) Selanjutnya kunjungi simpul tetangga dari simpul 4 (dalam kasus ini simpul tersebut sudah dikunjungi). Proses traversal berhenti di sini karena semua simpul sudah dikunjungi.

2.1.3. *Depth First Search* (DFS)

Depth First Search (DFS) adalah algortima pencarian sebuah node yang memenuhi kondisi tertentu pada struktur data graf atau *tree*. Proses pencarian dimulai dari simpul akar pohon dan menelusuri sejauh mungkin ke cabang tertentu sampai ke simpul daun. Kemudian mundur sampai menemukan simpul yang belum dikunjungi lalu jelahi seperti proses sebelumnya. Proses pencarian ini dilakukan sampai seluruh simpul sudah dikunjungi.

Proses traversal pada algortima DFS:
Misalkan proses traversal dimulai dari simpul v
a) Kunjungi simpul v .
b) Kunjungi simpul w yang bertetangga dengan simpul v v
c) Ulangi DFS dari simpul w .
d) 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.
e) Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Contoh traversal pada graf pada gambar 10:
Traversal dimulai dari simpul 0.
a) Kunjungi simpul 0.
b) Kunjungi simpul yang bertetangga dengan 0 yang memiliki angka terkecil, yaitu 1.
c) Kunjungi simpul yang bertetangga dengan 1 yang memiliki angka terkecil, yaitu 2. Simpul 0 tidak dipilih karena sebelumnya sudah dikunjungi.
d) Kunjungi simpul yang bertetangga dengan 2 yang memiliki angka terkecil, yaitu 4. Simpul 0 tidak dipilih karena sebelumnya sudah dikunjungi.
e) Karena simpul 4 merupakan simpul daun, *backtrack* ke simpul sebelumnya sampai ditemukan simpul yang bertetangga yang belum dikunjungi, yaitu simpul 3.
f) Karena simpul 3 merupakan simpul daun, lakukan *backtrack* ke simpul sebelumnya sampai ditemukan simpul yang bertetangga yang bertelum dikunjungi, semua simpul sudah dikunjungi.
g) Karena semua simpul sudah dikunjungi, proses traversal sudah selesai dilakukan.

2.2. C# *desktop application development*

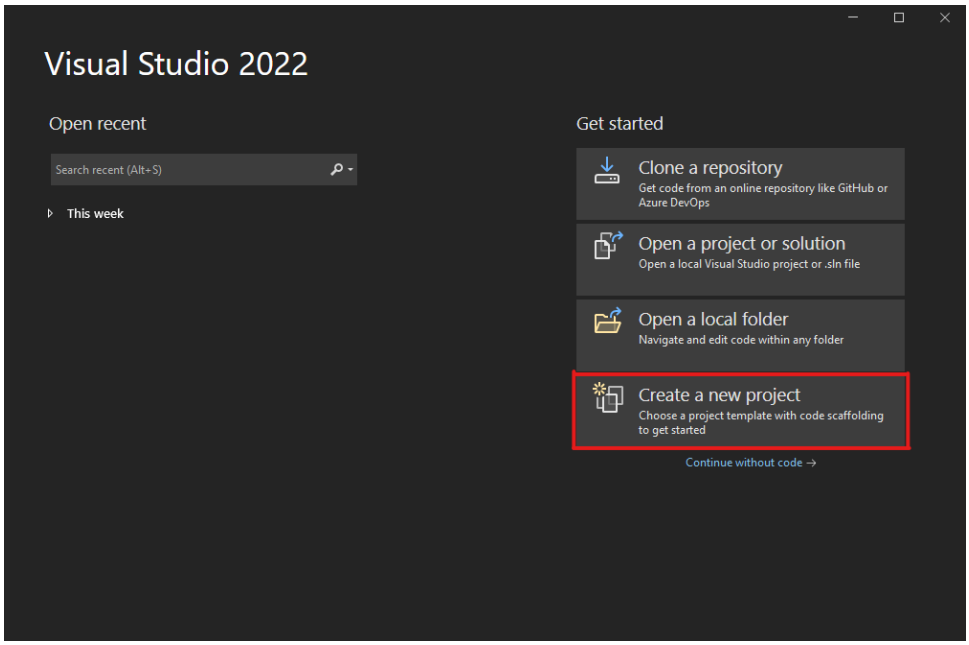
C# salah satu bahasa yang biasanya digunakan untuk pemograman *server-side website*, membangun aplikasi *desktop*, ataupun *mobile*, pemrograman, dan sebagainya. C# adalah bahasa yang tergolong baru dari Microsoft yang didesain semata-mata untuk menjalankan .NET framework. Bahasa ini dikembangkan dari bahasa C, C++, dan Visual Basic. Bahasa ini didesain sebagai bahasa pemrograman berorientasi objek yang sederhana dan modern. Komponen dari .NET secara garis besar dibagi menjadi tiga:
1) .NET Framework yang merupakan wadah pengembangan aplikasinya. .NET Framework juga dapat dibagi lagi menjadi tiga bagian, yakni
    a. CLR, sebuah *execution environment* yang mengatur memori alokasi, *error traping*, dan interaksi dengan sistem operasi.

b. Base Class Library, sebuah kumpulan komponen program beserta *application program interfaces* (APIs)nya.
c. Target development, yang pertama unuk *web applications* (ASP.NET) dan yang lain untuk *windows application* (Windows Form).

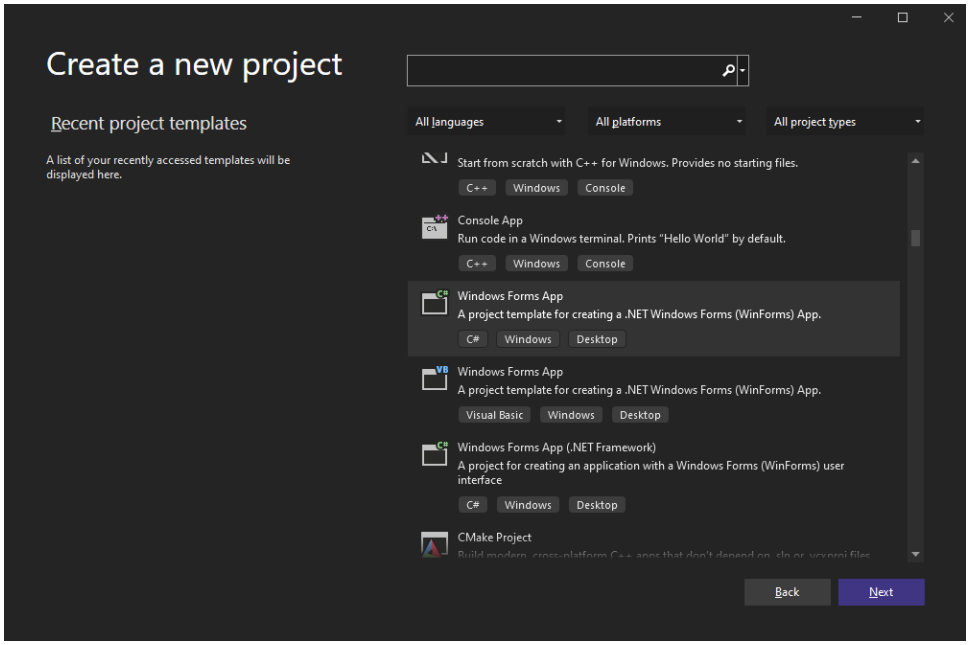
2) Produk-produk .NET, seperti beberapa aplikasi seperti Microsoft Exchange Server dan Microsoft SQL Server.
3) Layanan .NET yang disediakan oleh Microsoft dalam pengembangan aplikasi berbasis .NET Framework, misalnya Microsoft's Hailstorm projek yang merukan usaha Microsoft dalam membuat paket dari Web Services yang penting.

Untuk pengembangan aplikasi *desktop*, terdapat berbagai jenis aplikasi yang disediakan oleh Visual Studio antara lain Console Application, Windows Forms Application, WPF Aplication, dan sebagainya. Setiap jenis aplikasi memiliki kekurangan dan kelebihan 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. Adapun langkah-langkah untuk membuat Windows Forms Application dengan Visual Studio sebagai berikut:

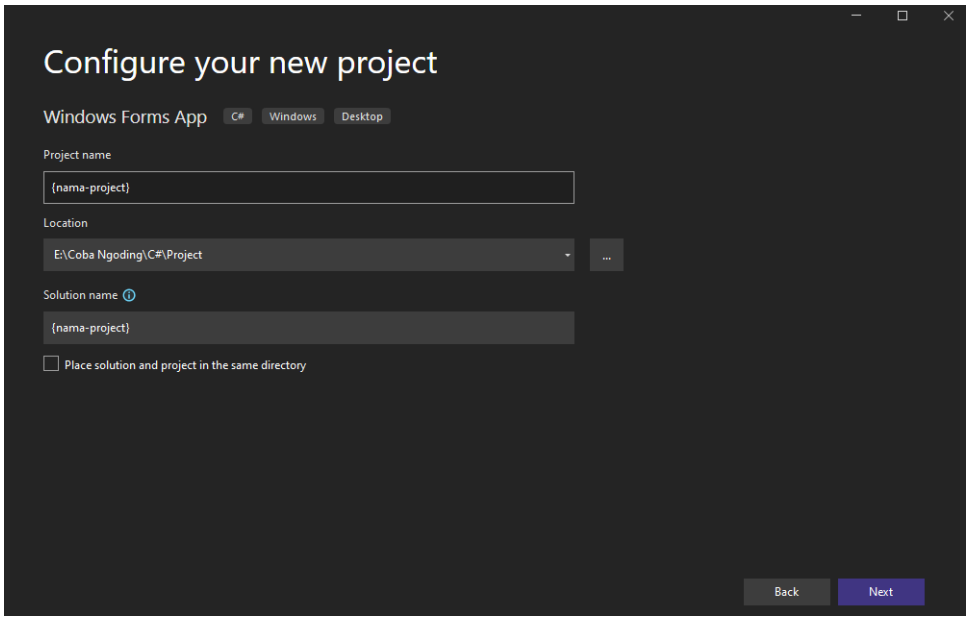
1. Unduh Visual Studio dari laman berikut https://visualstudio.microsoft.com/downloads/
2. *Install* dan jalankan Visual Studio yang telah diunduh sebelumnya.
3. Pada jendela yang terbuka, klik "Create a new project"



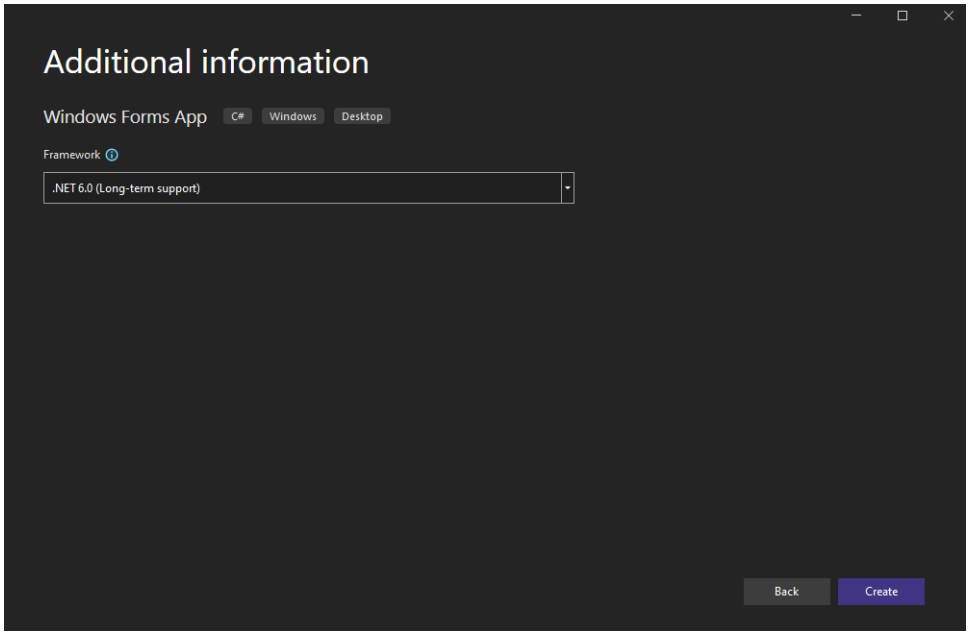
4. Pilih Windows Form App sebagai template dari aplikasi yang akan dibuat. Kemudian klik Next.



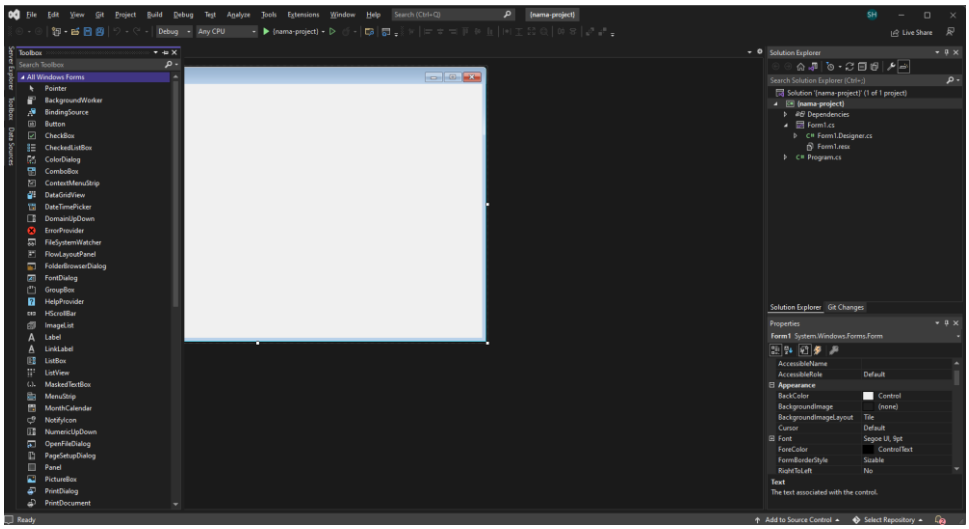
5. Pada jendela “Configure your new project”, tuliskan nama dari aplikasi yang akan dibuat dan tentukan lokasi penyimpanan dari aplikasi yang akan dibuat. Kemudian klik Next.



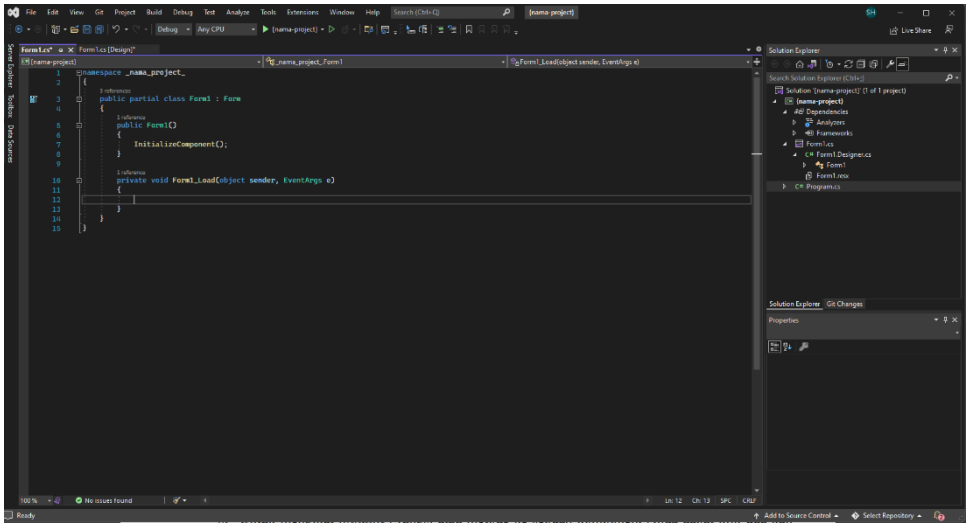
6. Pilih jenis Framework yang ingin digunakan. Disarankan untuk memilih Framework terbaru agar dapat menggunakan lebih banyak fitur yang disediakan oleh C# dan Visual Studio. Kemudian klik “Create”.



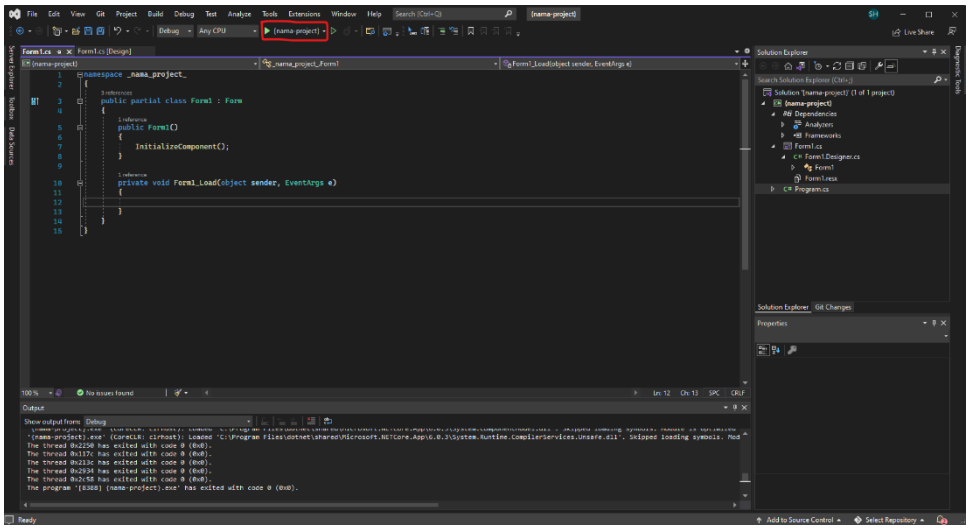
7. Akan ditampilkan form *default* dari Windows Form. Untuk menambahkan komponen pada Form, klik bar Toolbox di panel sebelah kiri. Tarik komponen yang ingin digunakan ke Form. Anda dapat berkreasi sebebas mungkin dengan komponen yang telah disediakan.

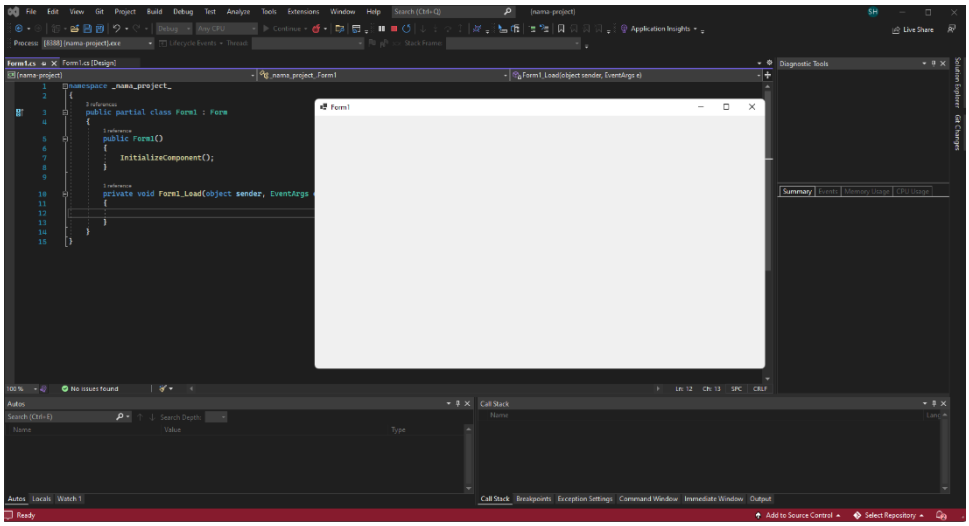


8. Untuk mengikat (*binding*) fungsi atau metode ke sebuah komponen, Anda dapat klik dua kali pada komponen yang ingin digunakan. Lalu akan secara otomatis diarahkan ke sebuah *method* pada Form1.cs



9. Untuk menjalankan aplikasi, klik pada logo “play” berwarna hijau diatas. Visual Studio secara otomatis akan melakukan *build* terhadap aplikasi yang dibuat. Jika *build* berhasil, aplikasi (.exe) akan dijalankan secara otomatis.





Bab 3
Analisis Pemecahan Masalah

3.1. Pemecahan Masalah

Pada *directory* yang dipilih, daftarkan semua *subdirectory* (folder) dan *file* yang ada di *directory* tersebut. Kemudian, periksa pada *file-file* di *directory* tersebut apakah sama dengan *file* yang dicari. Jika file ditemukan, program akan menampilkan *path* dari folder *parent*-nya. Sebaliknya, jika *file* tidak ditemukan, program akan memeriksa folder-folder berikutnya yang belum ditelusuri.

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

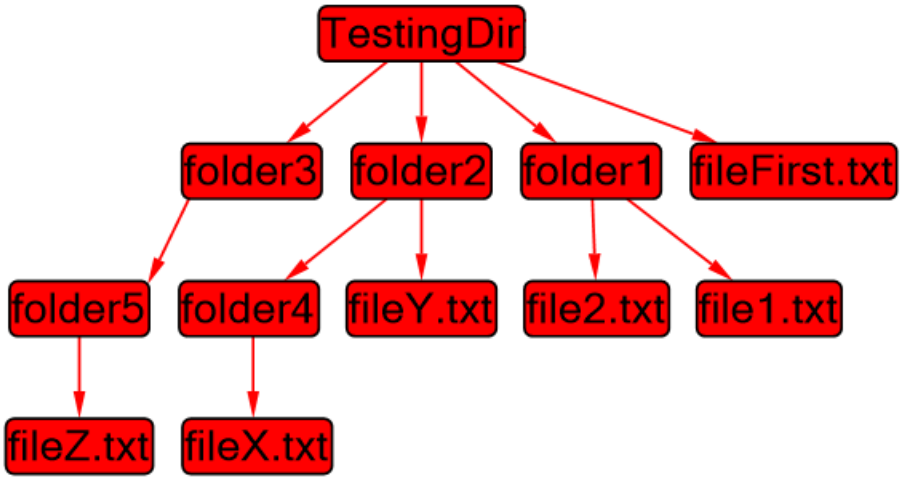
1) Dilakukan pengambilan nama *directory* dari *directory* yang dipilih oleh pengguna beserta nama *file* dari isian pengguna.
2) Nama *directory* dan *file* yang ditemukan dijadikan parameter dalam metode DFS dan BFS.
3) Pada metode DFS dan BFS, dilakukan pembentukan graf yang terdiri dari *node* dan *edge* terus dibentuk seiring pencarian *file*.
    a. Pada kasus *find all occurrences,* Semua *folder* ditelusuri dan setiap *file* yang sesuai disimpan *path*-nya dalam sebuah *list*
    b. Pada kasus *find first occurrence*, ketika *file* pertama ditemukan, pencarian dihentikan pada level tersebut.

3.3. Kasus Lain di luar Spesifikasi Tugas

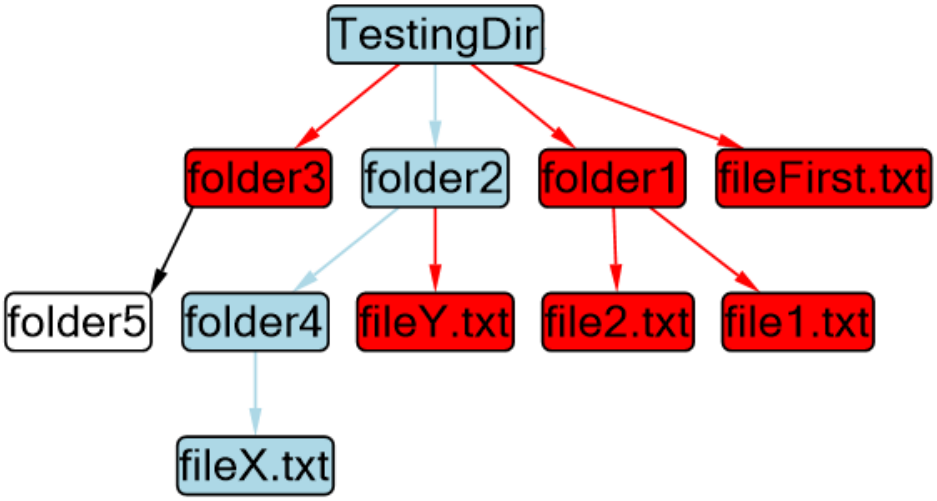
Contoh konten folder pengujian :

```
TestingDir/
    folder1/
        file1.txt
        file2.txt
    folder2/
        folder4/
            fileX.txt
        fileY.txt
    folder3/
        folder5/
        fileZ.txt
    fileFirst.txt
```

Visualisasi graf yang dihasilkan dari konten folder pengujian



Untuk pencarian fileX.txt dengan BFS, maka output yang diharapkan sebagai berikut.

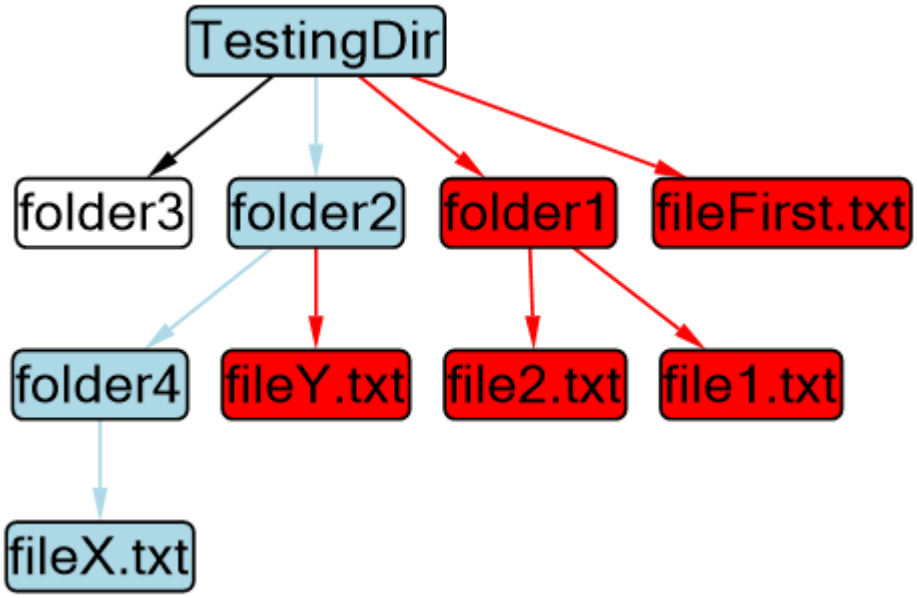


Adapun, path pencarian BFS adalah sebagai berikut.

TestingDir → fileFirst.txt → folder1 → folder2 → folder3 → file1.txt → file2.txt → fileY.txt → folder4 → fileX.txt

Pada gambar di atas, rute yang dilewati pada pencarian BFS diwarnai dengan warna merah. Sedangkan, rute untuk menuju tempat file berada diberi warna biru. Rute yang masuk antrian tapi belum diperiksa diberi warna hitam.

Untuk pencarian fileX.txt dengan DFS, maka output yang diharapkan sebagai berikut.

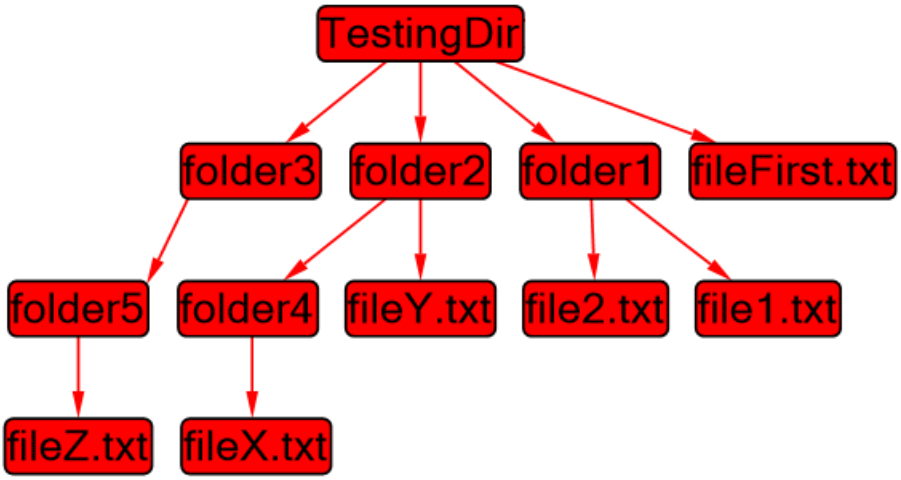


Adapun, path pencarian DFS adalah sebagai berikut.

TestingDir → fileFirst.txt → folder1 → file1.txt → file2.txt → folder2 → fileY.txt → folder4 → fileX.txt

Pada gambar di atas, rute yang dilewati pada pencarian BFS diwarnai dengan warna merah. Sedangkan, rute untuk menuju tempat file berada diberi warna biru. Rute yang masuk antrian tapi belum diperiksa diberi warna hitam.

Untuk pencarian fileA.txt dengan DFS ataupun BFS, maka output yang diharapkan sebagai berikut.



Adapun, path pencarian DFS adalah sebagai berikut.

TestingDir → fileFirst.txt → folder1 → file1.txt → file2.txt → folder2 → fileY.txt → folder4 → fileX.txt → folder5 → fileZ.txt

Kemudian, path pencarian BFS adalah sebagai berikut.

TestingDir → fileFirst.txt → folder1 → folder2 → folder3 → file1.txt → file2.txt → fileY.txt → folder4 → fileX.txt → folder5 → fileZ.txt

Pada gambar di atas, semua simpul dan cabang berwarna merah yang menandakan seluruh direktori sudah selesai diperiksa semua, tetapi tidak ada yang mengarah ke tempat file berada.

Bab 4
Implementasi dan Pengujian

4.1.Implementasi Program

4.1.1. *Pseudocode* BFS

```
WaitingList : array of string

function searchAll(selectedDir: string, file: string) → array of string
KAMUS
    paths, dirs, files, waitingList  : array of string
    prioDir : string
    found : boolean
    i : integer
ALGORITMA
    dirs ← list of string of directories in selectedDir
    files ← list of string of files in selectedDir
    found ← false


    i ← 0

    while (not found and i < files.Length) do
        if (file = files[i]) then
            paths.Add(selectedDir)
            found ← true
        i ← i + 1

    waitingList.Concat(dirs)
    if (waitingList.Count() ≠ 0) then
        priodir ← waitingList[0]
        paths.Concat(searchAll(prioDir, file))
    → paths


function searchOne(selectedDir: string, file: string) → string
KAMUS
    path, prioDir: string
    dirs, files: array of string
    found: boolean
    i: integer
ALGORTIMA
    dirs ← list of string of directories in selectedDir
    files ← list of string of files in selectedDir
    found ← false


    i ← 0
    while (not found and i < files.Length) do
        if (file = files[i]) then
            path = files[i]
            found = true
        i ← i + 1

    waitingList.Concat(dirs)
    if (waitingList.Count() ≠ 0 and not found) then
        priodir ← waitingList[0]
```

```
        waitingList.RemoveAt(0)
        path ← searchOne(prioDir, file)

    → path
```

4.1.2. *Pseudocode* DFS

```
function searchAll(selectedDir: string, file: string) → array of string
KAMUS
   paths, dirs, files : array of string
   found : boolean
   i : integer
ALGORITMA
   dirs ← list of string of directories in selectedDir
   files ← list of string of files in selectedDir
   found ← false

   i ← 0
   while (not(found) and i < files.Length) do
      if (file = files[i]) then
         paths.Add(selectedDir)
         found ← true
      i ← i + 1

   i traversal[0..dirs.Length-1]
      path.Concat(searchAll(dirs[i], file))
      → paths

function searchOne(selectedDir: string, file: string) → string
KAMUS
   path : string
   dirs : array of string
   files : array of string
   found : boolean
   i : int
ALGORITMA
   dirs ← list of string of directories in selectedDir
   files ← list of string of files in selectedDir
   found ← false

   i ← 0
   while (not(found) and i < files.Length) do
      if (file = files[i]) then
         path ←←selectedDir
         found ← true
      i ← i + 1

   i ← 0
   while (not(found) and i < dirs.Length) do
      path ← searchOne(dirs[i], file);
      if (path is not empty) then
         found ← true
      i ← i + 1

   → path
```

4.2. Struktur Data

4.2.1. Kelas pada library MSAGL

Terdapat kelas graph dan viewer yang disediakan. Graph sendiri memiliki subkelas *node* dan *edge*, sedangkan *viewer* merupakan *class* untuk menampilkan hasil *graph*-nya.

4.2.2. Kelas BFS

Kelas BFS merupakan implementasi dari alggoritma BFS. Kelas ini mempunyai atribut waitingList yang menyimpan simpul-simpul atau folder yang akan ditelusuri. Selain itu, pada kelas ini terdapat *method* searchAll, untuk mencari semua file yang namanya sama dengan file yang dicari, dan *method* searchOne, untuk mencari hanya file yang pertama kali ditemukan dan namanya sama dengan file yang dicari. Masing-masing *method* tersebut menerima parameter selectedDir yang merupkan *directory* yang akan ditelusuri dan file yang merupakan nama file yang ingin dicari.

4.2.3. Kelas DFS

Kelas DFS merupakan implementasi dari algoritma DFS. Kelas ini mempunyai *method* searchAll, untuk mencari semua file yang namanya sama dengan file yang dicari, dan *method* searchOne, untuk mencari hanya file yang pertama kali ditemukan dan namanya sama dengan file yang dicari. Masing-masing *method* tersebut menerima parameter selectedDir yang merupakan *directory* yang akan ditelusuri dan file yang merupakan nama file yang ingin dicari.

4.3. Pengunaan Program

4.3.1. *Interface* yang disediakan

1) *Button choose folder* untuk memilih *directory* mulai pencarian
2) *Textbox input* file untuk memasukan file
3) *Checkbox* validasi jenis lingkup pencarian
4) *RadioButton* pemilihan jenis algoritma (BFS atau DFS)
5) *Button search* untuk memulai proses pencarian
6) *Panel* untuk menampilkan *graf* hasil pencarian
7) Label statis untuk deskripsi tiap bagian pada *form*
8) Label dinamis, yakni label *time spent* dan label *hyperlink* yang akan muncul jika pencarian berhasil

4.3.2. Fitur program

1) Menampilkan *graf* pencarian
2) Menghasilkan *hyperlink* untuk pengguna bisa langsung mengakses *folder* tempat file ditemukan jika diklik
3) Menampilkan *time spent* proses pencarian

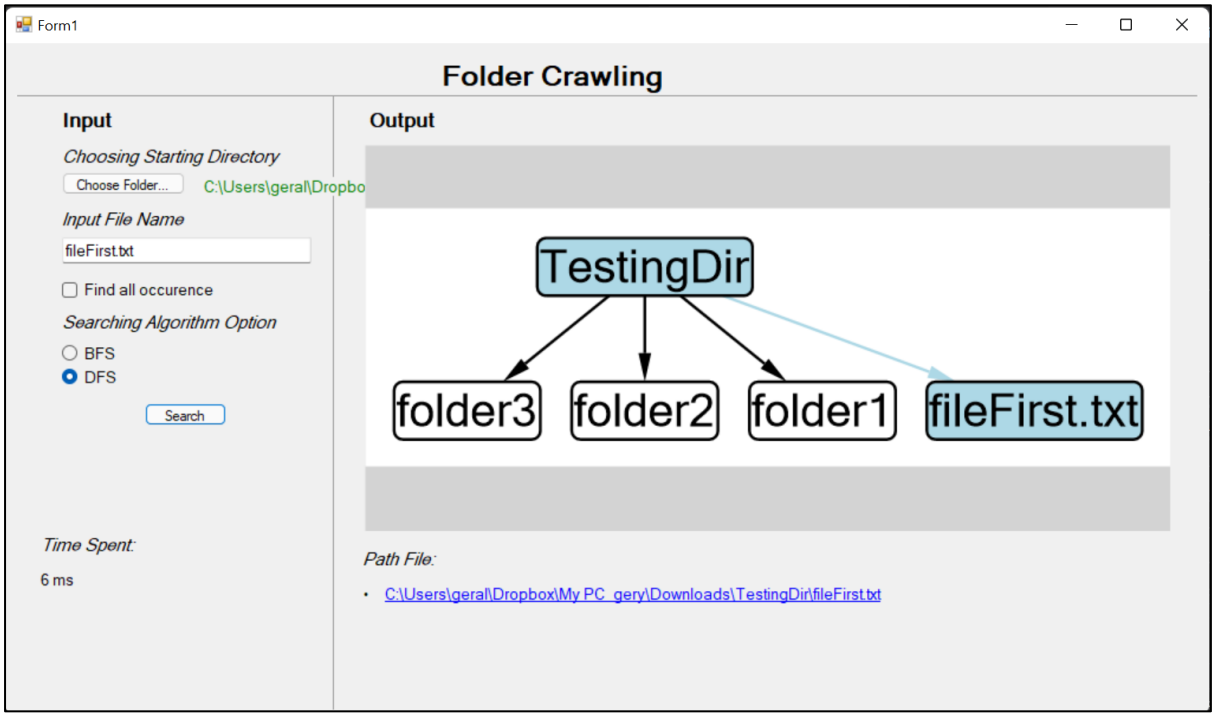
4.4. Pengujian

4.4.1. Konten *folder* pengujian

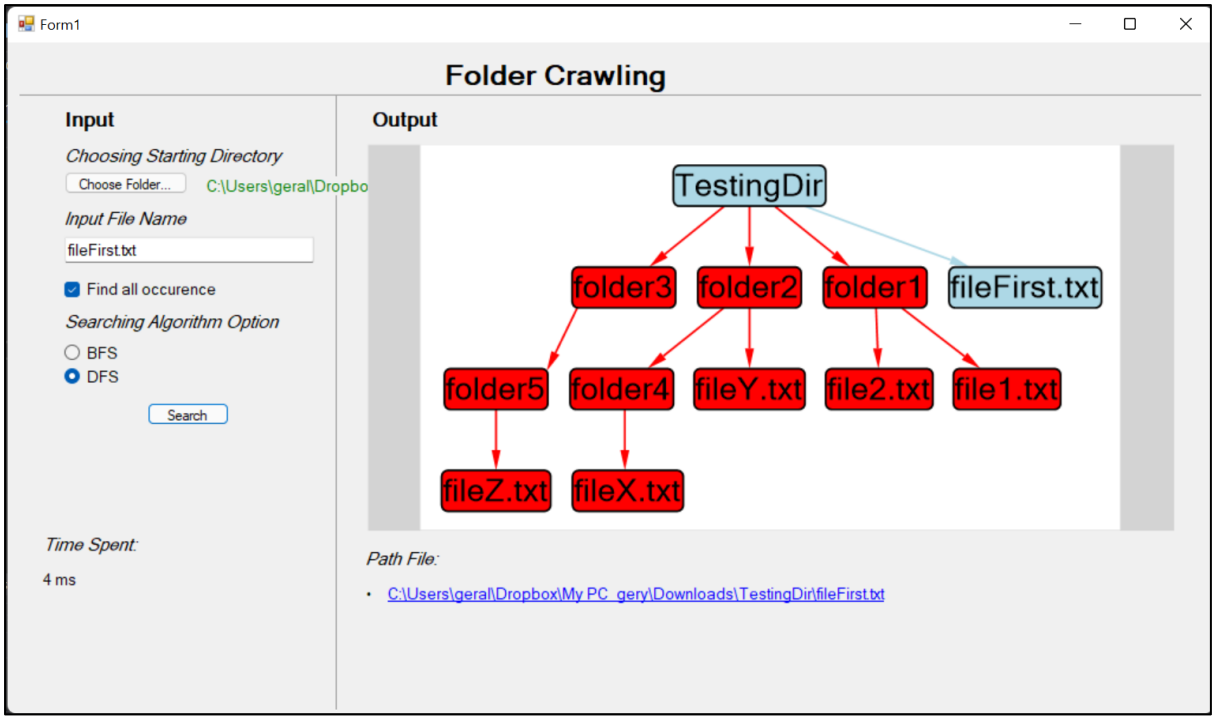
```
TestingDir/
    folder1/
        file1.txt
        file2.txt
    folder2/
        folder4/
            fileX.txt
        fileY.txt
    folder3/
        folder5/
        fileZ.txt
    fileFirst.txt
```

4.4.2. Hasil pengujian

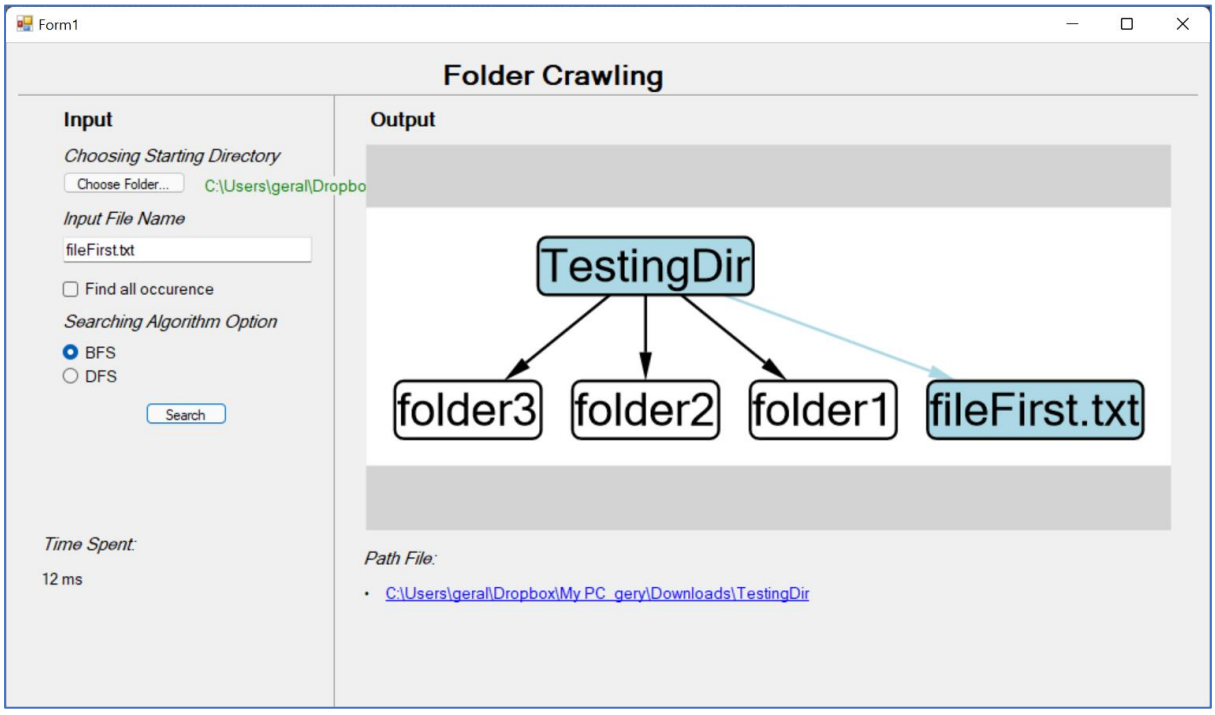
1) Pencarian *file* yang *exist*
    a. *file* di level 1 (fileFirst.txt)
        i. DFS (*first occurrence*)



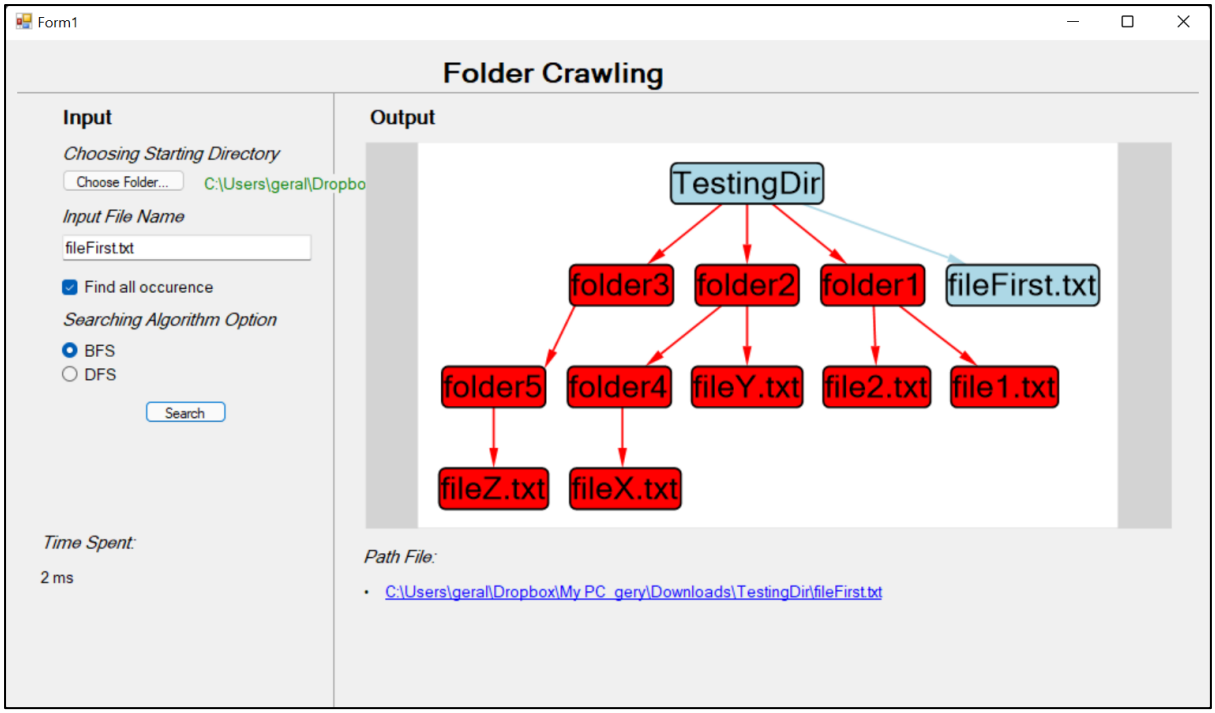
        ii. DFS (*All occurrences*)



iii. BFS (*first occurrence*)

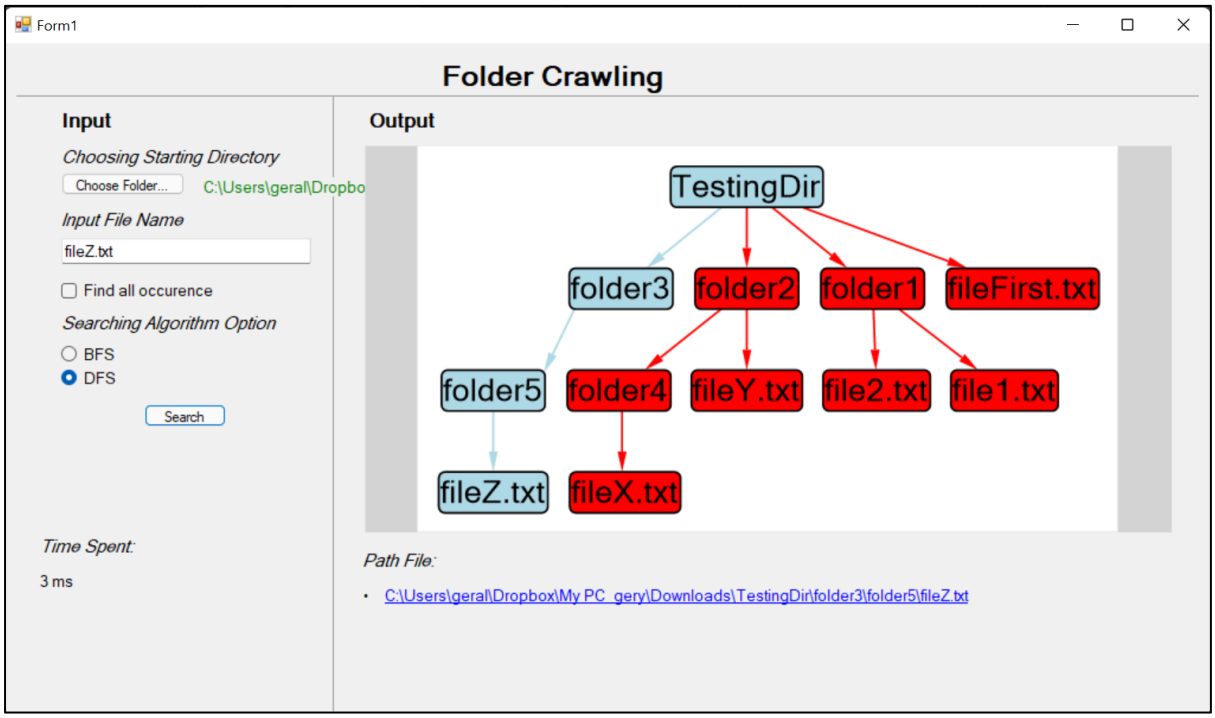


iv. BFS (*All occurrences*)

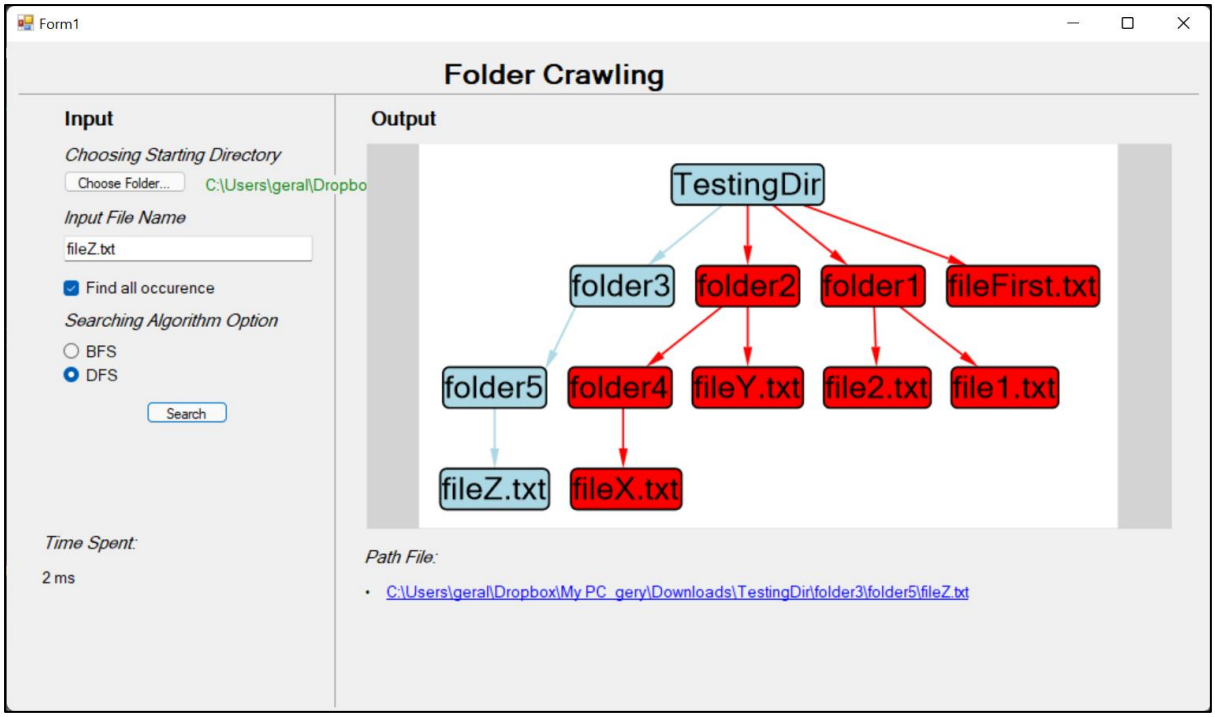


b. *file* selain di level 1 (fileZ.txt)

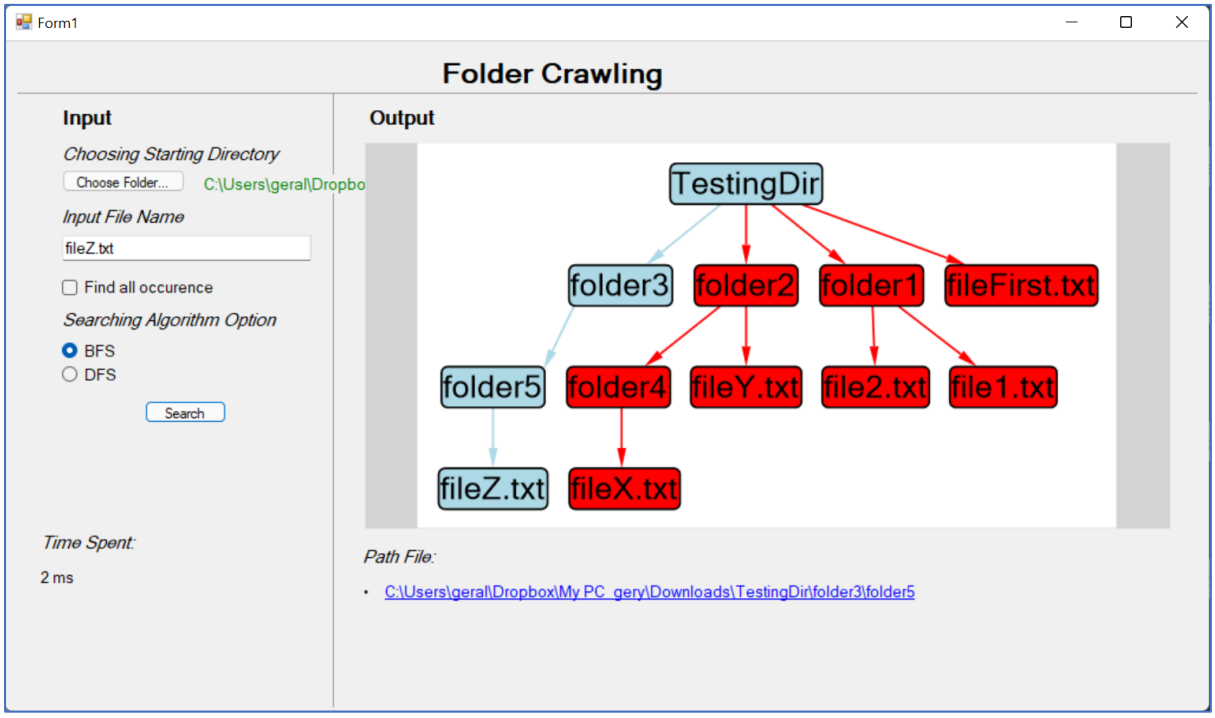
i. DFS (*first occurrence*)



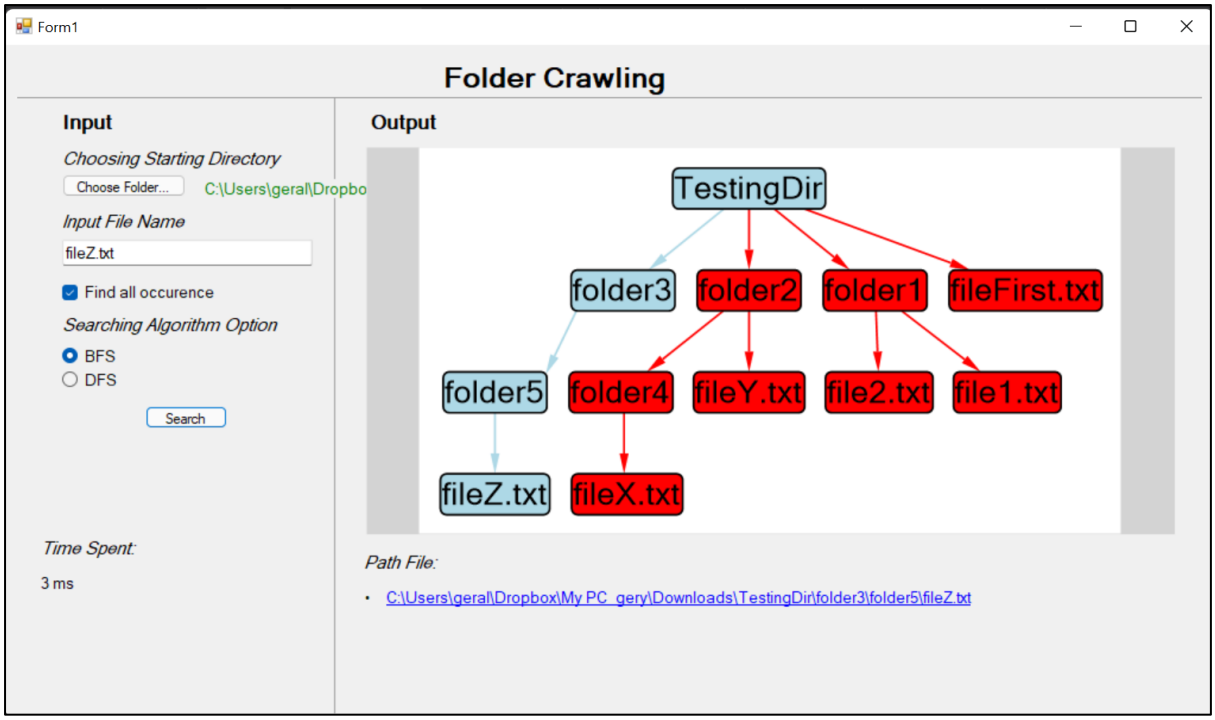
ii. DFS (*All occurrences*)



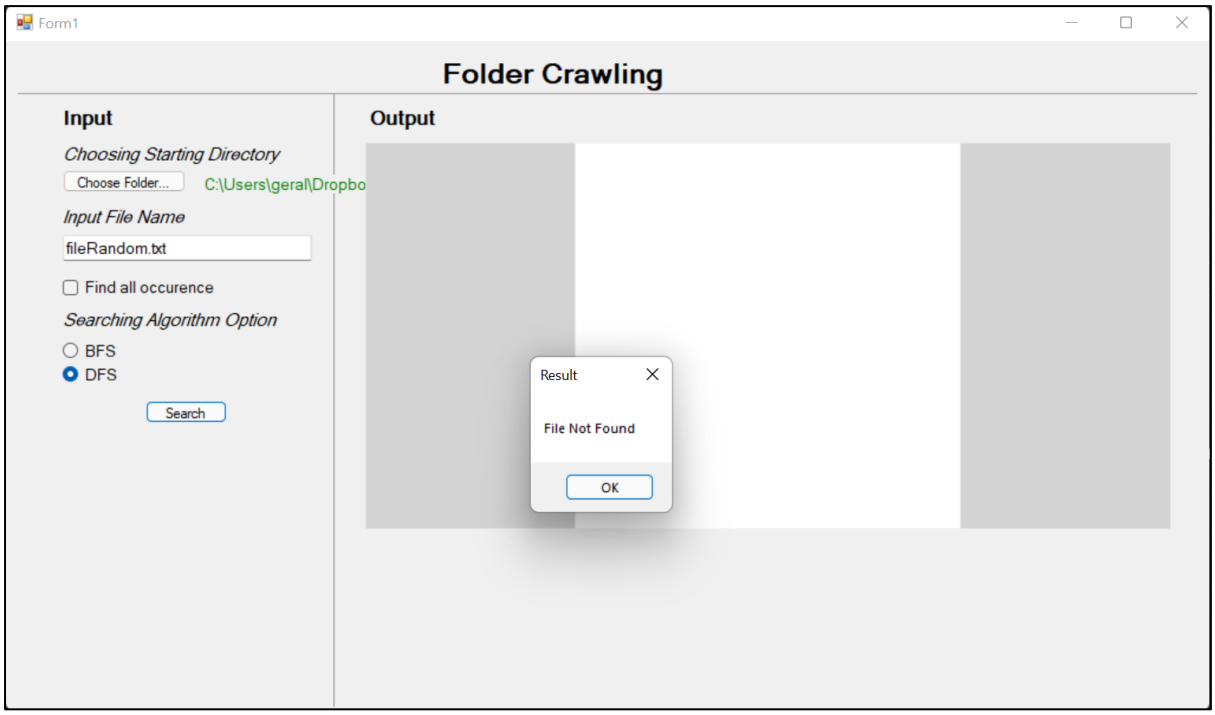
iii. BFS (*first occurrence*)

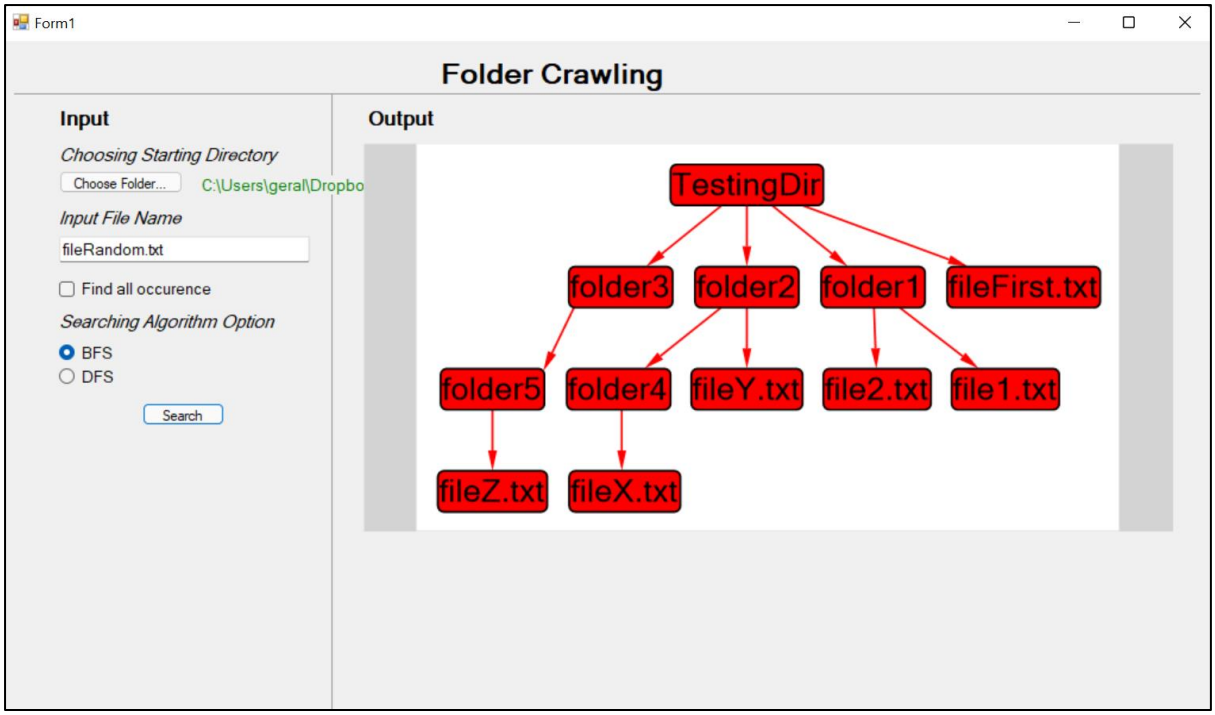


iv. BFS (*All occurrences*)



2) Pencarian *file* yang tidak ada (fileRandom.txt)
   a. BFS (*first occurence*)

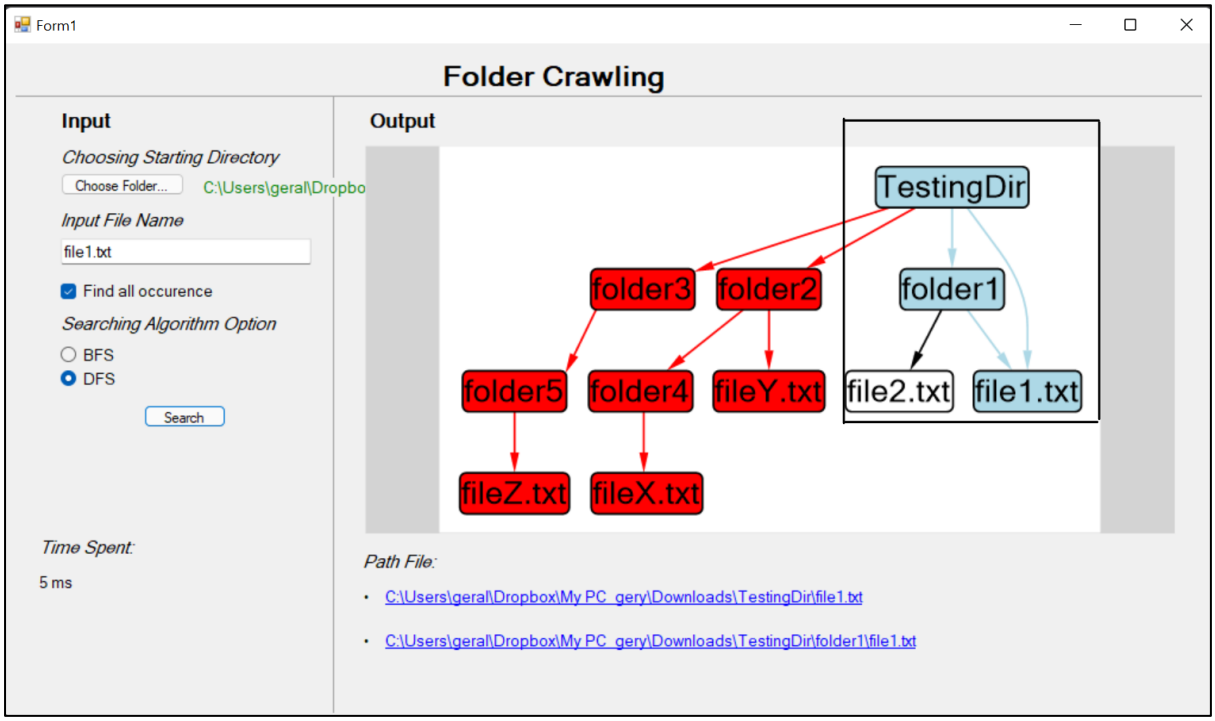




b. DFS (*all occurences* dan *first occurence*) dan BFS (*first occurrence*) → tampilan sama seperti kasus di atas (kasus 2a)

4.4.3. Temuan hasil yang tidak diharapkan (*bug*)

Program yang kami buat belum meng-*handle* kasus yang memiliki file sama dengan baik, dalam artian tidak akan terbentuk *node* baru jika ada file yang sama sebelumnya. Contoh, fileFirst.txt diubah menjadi file1.txt, dan dilakukan pencarian DFS (all occurences). Hasil yang ada sebagai berikut:



Pada kasus ini, pertama, ter-*generate edge* (TestingDir → file1.txt), kemudian ter-*generate edge* (folder1 → file1.txt), folder1 akan menunjuk ke *node* yang sama pada file1.txt sebelumnya.

4.5.Analisis Desain Solusi

Perbedaan mendasar dari BFS dan DFS adalah sesuai dengan namanya. DFS mencari ke kedalaman *node* yang masih dapat ditelusuri. Sementara BFS melakukan iterasi ke setiap lapisan *node* baru yang dikunjungi. Struktur data paling umum yang dipakai DFS adalah *stack* karena sebagaimana *stack* menerapkan *first in last out.* Sedangkan struktur data paling umum yang dipakai BFS adalah *queue* karena sebagaimana *queue* menerapkan *first in first out*. Kasus-kasus dimana DFS dan BFS mengungguli atau diungguli yang lain dapat kita bedah lebih lanjut dengan penjabaran keuntungan masing-masing algoritma berikut.

Kuntungan BFS dibanding DFS adalah bahwa BFS melakukan iterasi per lapisan, sehingga dalam kasus *mono-weighted graph*, BFS dijamin menghasilkan jalur terpendek. Fakta uniknya, penelusuran BFS pada *mono-weighted graph* sama persis dengan Algoritma Djikstra. Jika dilanjutkan hingga Queue kosong, BFS dapat menghitung jarak minimal tiap node pada graph dengan node awal. Lebih lanjutnya lagi, dapat dilakukan multisource BFS yakni BFS dengan node awal lebih dari satu untuk menyelesaikan permasalahan tertentu. Kompleksitas waktu dari BFS ialah jumlah edge dari connected-component tempat node asal berada, sedangkan kompleksitas memorinya ialah jumlah node dalam connected-component node asal. Connected component ialah node pada graph yang saling terhubung.

Selanjutnya DFS juga terkadang lebih menguntungkan dari BFS untuk sedikit kasus misalnya saat graph berbentuk tree, misalkan node awal merupakan akarnya, jumlah subpohon dari akar cukup banyak, dan node yang dituju berada pada subpohon-subpohon awal. Maka BFS akan membuang waktu mencari pada lapisan demi lapisan yang jumlahnya dapat bertambah banyak secara eksponensial, sementara DFS akan mencari satu subpohon dulu secara menyeluruh. Kerugian DFS yakni jalur yang dihasilkan tidak selalu minimal tentunya berpengaruh pada pengecekan node awal dan yang dituju sudah terhubung dengan edge atau belum. Namun disamping itu, DFS dapat diimplementasikan dengan metode rekursi yang sangat pendek dan mudah dalam proses pengembangan kode.

Meskipun BFS jauh lebih disarankan dalam pathfinding, DFS dapat melakukan hal yang tidak dapat dilakukan BFS, misalnya pada kasus Dynamic Programming (DP) on Tree, konstruksi Bridge Tree, pencarian Articulation Point, Biconnected Component/Strongly Connected Component (BC/SCC), dan lainnya. DP on Tree ialah struktur data yang menyimpan banyaknya node pada subtree dari masing-masing node. BC/SCC ialah connected component yang bila dihapus satu edge manapun akan tetap terhubung. Articulation Point ialah node yang jika dihapus akan membuat graph menjadi tidak terhubung. Terakhir Bridge Tree ialah pohon yang node nya berupa BC/SCC pada graph, dan node nya adalah tiap edge pada graph yang jika dihapus akan membuat graph tidak terhubung.

Bab 5
Kesimpulan dan Saran

5.1. Kesimpulan

Berdasarkan yang sudah dipaparkan sebelumnya, dapat diambil kesimpulan bahwa struktur data graf dapat merepresentasikan folder crawling dengan baik karena memudahkan dalam proses traversal dari atribut-atributnya. Proses traversal yang dimaksud adalah metode pencarian DFS dan BFS.

Metode pencarian BFS dan DFS pada kasus mencari direktori yang diinginkan dapat menemukan solusi yang baik dan tidak jauh berbeda. Namun, karena terdapat perbedaan cara penelusuran *node*, maka jalur yang dilalui saat proses traversal berbeda dan metode BFS cenderung menghasilkan solusi jalur yang terpendek. Dalam hal ini, program kami sudah dapat menerapkan metode pencarian BFS dan DFS dalam fitur dari file explorer pada sistem operasi, dengan tepat.

5.2. Saran

Saran-saran yang dapat kami berikan untuk tugas besar IF 2211 Strategi Algoritma semester 2 2021/2022 adalah:

a. Algoritma yang digunakan pada Tugas Besar ini masih memiliki banyak kekurangan sehingga sangat memungkinkan untuk dilakukan efisiensi, misalnya dengan tidak menggunakan fungsi yang sama berulang-ulang. Oleh karena itu, dalam pengembangan program ini, masih bisa dilakukan efisiensi kerja.
b. Memperjelas spesifikasi dan batasan-batasan setiap program pada *file* tugas besar untuk mencegah adanya multitafsir dan kesalahpahaman pada proses pembuatan program.
c. Penulisan *pseudocode* tampak kurang perlu dikarenakan program yang lumayan panjang dan membaca program lebih mudah daripada membaca *pseudocode* dengan asumsi program sudah *well commented*.

5.3. Refleksi dan Komentar

Pada pengerjaan Tugas Besar 2 Strategi Algoritma ini, kami sadar akan pentingnya menentukan prioritas dan pembagian kerja yang baik saat bekerja dalam sebuah tim karena pada dunia nyata setiap anggota tim memiliki kesibukan atau tuntutan lain diluar tugas besar ini. Faktor penting lainnya adalah komunikasi dan transparansi antaranggota. Hal ini akan meminimalisir kemungkinan terjadinya miskomunikasi dan memudahkan untuk *keep in track progress* dari yang sudah dikerjakan oleh masing-masing anggota sehingga dapat mengatasi masalah lebih cepat dan taktis. Diluar dari hal yang telah disebutkan, yang terpenting adalah untuk selalu melatih keterampilan diri sendiri secara rutin dan berkala serta mengeksplor pengetahuan-pengetahuan yang tidak diajarkan dalam kelas sehingga dapat lebih adaptif saat dihadapkan pada suatu permasalahan.

Link repository github:

https://github.com/geraldabrhm/Stima02.git