Laporan Tugas Besar 2 IF 2211

Strategi Algoritma

Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling

Disusun Oleh:

ParaPencariFile

Rizky Akbar Asmaran	**13520111**
Muhammad Alif Putra Yasa	**13520135**
Haidar Ihzaulhaq	**13520150**

Institut Teknologi Bandung

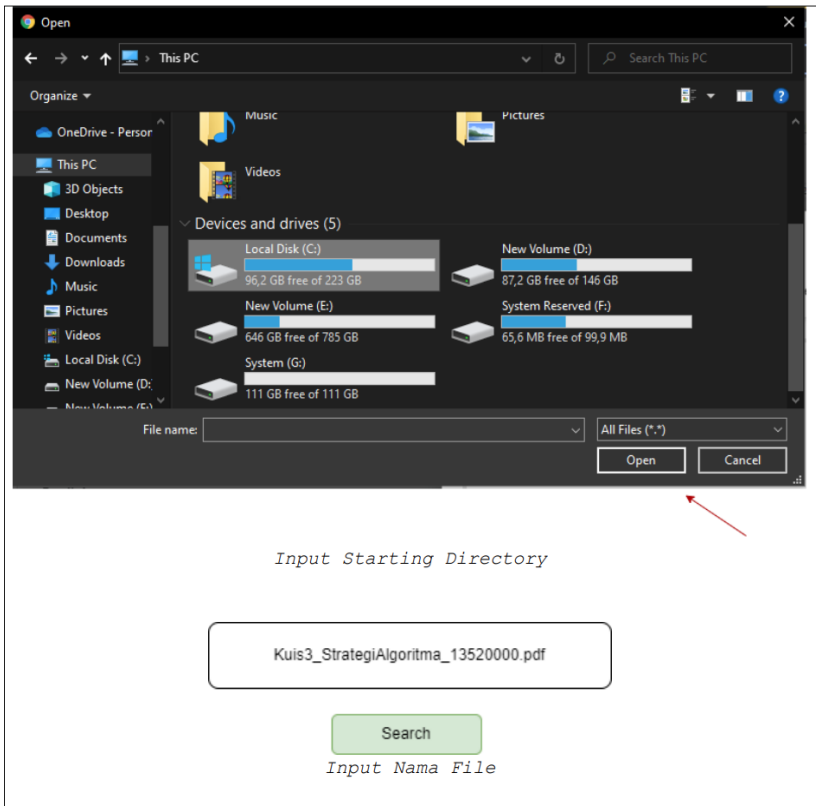
Bandung

2022

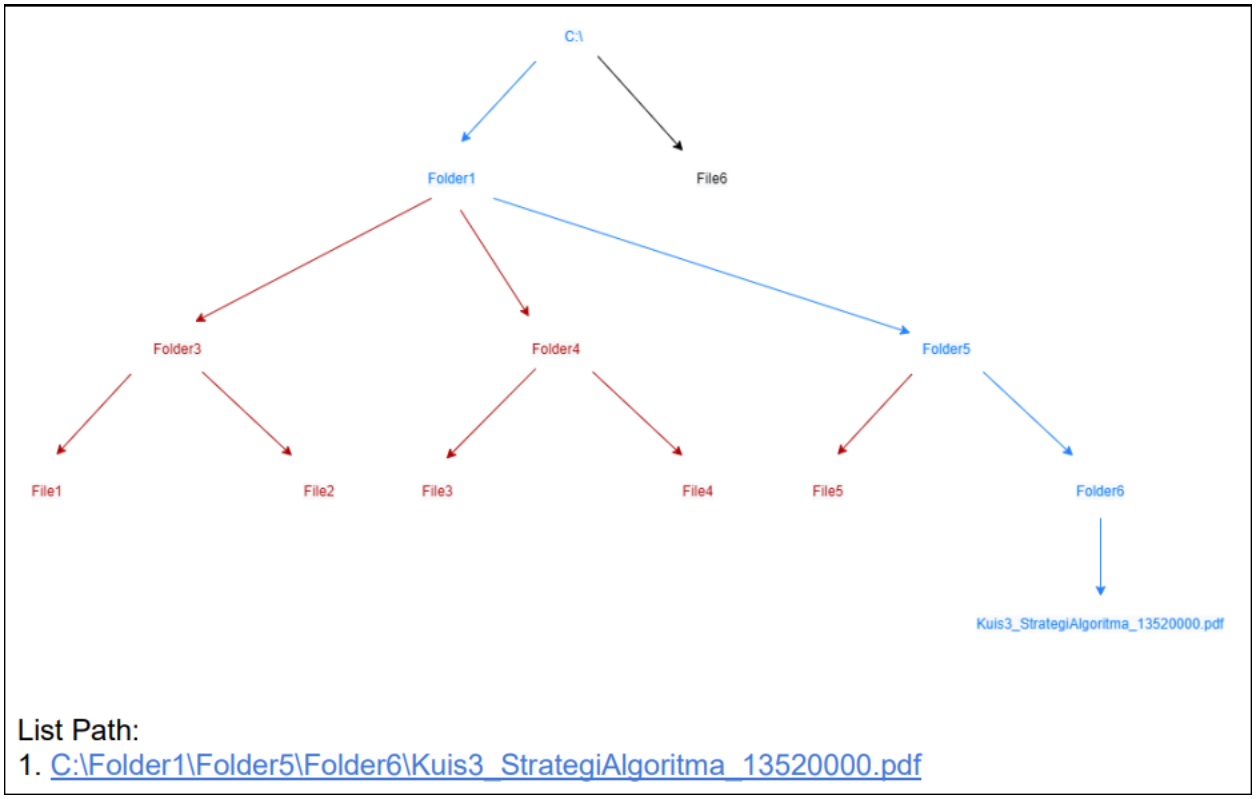
BAB I
DESKRIPSI TUGAS

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.

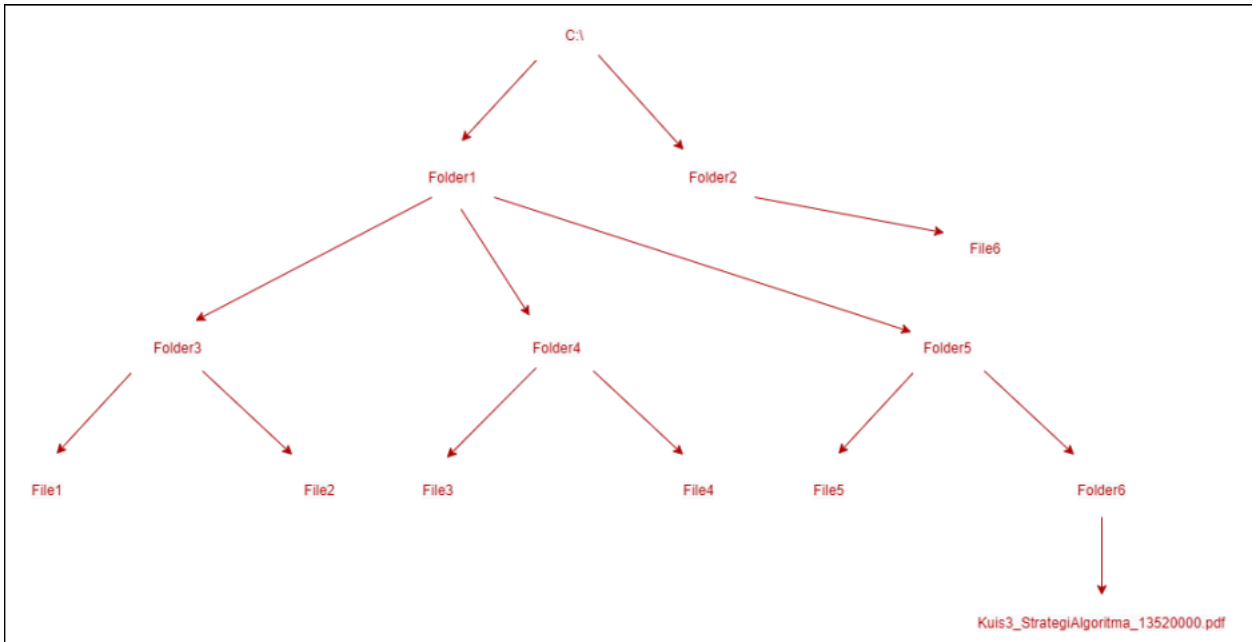
Contoh Input dan Output Program Contoh masukan aplikasi:



Contoh output aplikasi:



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.



List Path:
Tidak ada path yang sesuai.

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.

BAB II
LANDASAN TEORI

2.1 Graph Traversal

Graf adalah sebuah representasi persoalan dan traversal graf adalah pencarian solusi. Jadi, Graf traversal adalah suatu proses yang mengacu pada proses mengunjungi tiap simpul dalam graf dengan cara yang sistematik, terdapat beberapa macam cara pengunjungan yang dapat dilakukan, yaitu dengan Pencarian melebar (*Breadth First Search / BFS*) DAN Pencarian Mendalam (*Depth First Search / DFS*) dengan asumsi graf terhubung.

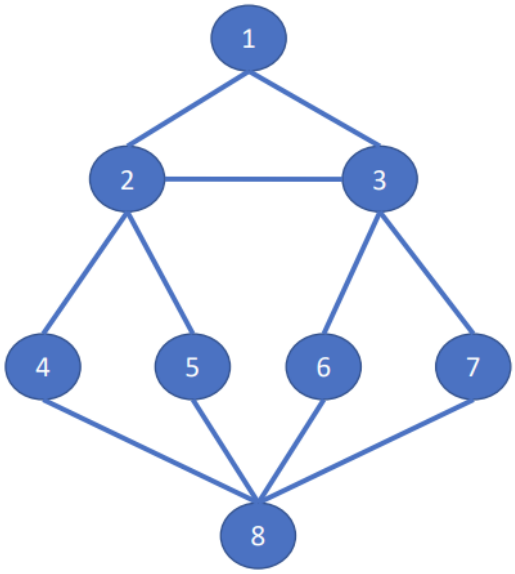
Pada Graf Traversal ini digunakan Algoritma Pencarian Solusi dengan 2 jenis, pertama, tanpa informasi (*uninformed* / blind search) yaitu tidak adanya informasi tambahan contohnya DFS, BFS, *Depth Limited Search, Iterative Deepening search, Uniform Cost Search*. Kedua, dengan informasi (*Informed Search*), dengan informasi yaitu dengan melakukan pencarian berbasis heursitik dengan mengetahui non-goal state "lebih menjanjikan" daripada yang lain.

Dalam proses pencarian solusi, terdapat dua buah pendekatan, pertama, graf statis, graf yang sudah terbentuk sebelum proses pencarian dilakukan, graf direpresentasikan sebagai struktur data. Kedua, graf dinamis, graf yang sudah terbentuk saat proses pencarian dilakukan dimana graf tidak tersedia sebelum pencarian dan graf dibangun selama pencarian solusi.

a. BFS

Breadth First Search atau pencarian melebar adalah suatu proses traversal yang dimulai dari simpul v. pada BFS ini dilakukan beberapa tahapan algoritma sebagai berikut:

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



BFS juga memiliki struktur data sebagai berikut:

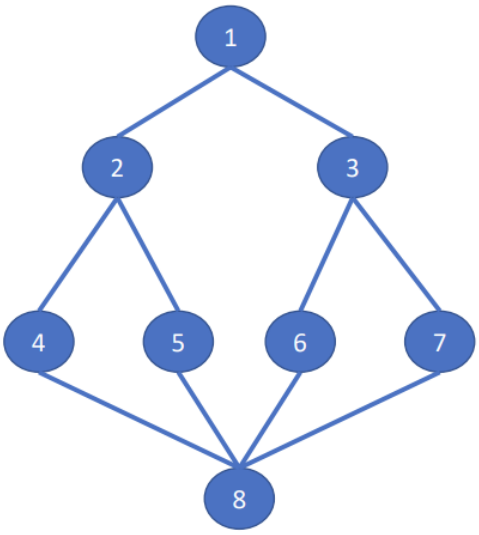
- Matriks ketetanggaan $A = [a_{ij}]$ yang berukuran *nxn*, jika simpul i dan simpul j bertetangga, $a_{ij} = 1$. Jika simpul i dan j tidak bertetangga, $a_{ij} = 0$.
- Antrian q untuk menyimpan simpul yang telah dikunjungi
- Tabel Boolean, diberi nama "dikunjungi"

  Dikunjungi : array[1..n] of boolean

```
Procedure BFS (input v: integer)
{ Traversal graf dengan algoritma pencarian BFS
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi dicetak ke layar
}

Deklarasi
  w: integer
  q: antrian,

Procedure BuatAntrian (input/output q : antrian)
{membuat antrian kosong, kepala(q) diisi 0}

Procedure MasukAntrian (input/output q: antrian, input v: integer)
{memasukkan v ke dalam antrian q pada posisi belakang}

Procedure HapusAntrian (input/output q: antrian, output v: integer)
{menghapus v dari kepala antrian q}
```

```
Function AntrianKosong (input q: antrian) -> boolean
{ true jika antrian q kosong, false jika sebaliknya }

ALGORITMA

BuatAntrian(q)              {buat antrian kosong}

write(v)                    {cetak simpul awal yang dikunjungi}
Dikunjungi [v] <- true      {simpul v telah dikunjungi, tandai dengan true}

MasukAntrian(q,v)           {masukan simpul awal kunjungan ke dalam
antrian}

{kunjungi semua simpul grad selama antrian belum kosong}

While not AntrianKosong(q) do
   HapusAntrian(q,v)        {simpul v telah dikunjungi, hapus dari antrian}

   For tiap simpul w yang bertetangga dengan simpul v do
         If not dikunjungi[w] then
            write(w)
            MasukAntrian(q,w)
            Dikunjungi[w] <- true
         Endif
   Endfor
Endwhile
{AntrianKosong(q)}
```

Ilustrasi:



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

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

b. DFS

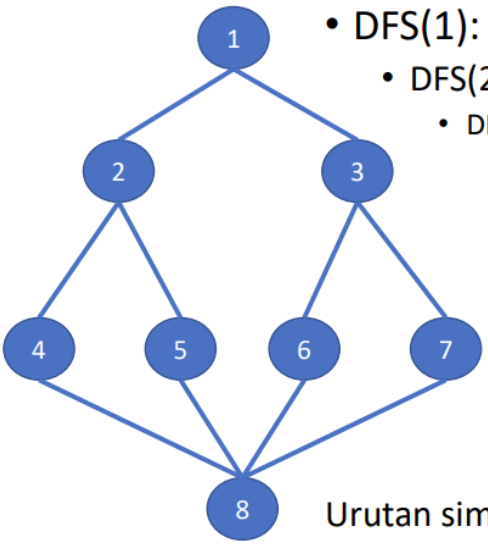
Depth First Search atau Pencarian Mendalam adalah suatu proses pencarian traversal yang dimulai dari simpul v. berikut Algoritmanya

- Kunjungi simpul v
- Kunjungi simpul w yang bertetangga dengan simpul v
- Ulangi DFS mulai dari simpul w
- 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.
- Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

```
Procedure BFS (input v: integer)
{ Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS

masukan : v adalah simpul awal kunjungan
Keluaran : semua simpul yang dikunjungi ditulis ke layar
}

Deklarasi
  w: integer
```

```
ALGORITMA

   write(v)
   Dikunjungi[v] <- true
   For w <- 1 to n do
        If A[v,w] = 1 then {simpul v dan simpul w bertetangga}
           If not dikunjungi[w] then
                DFS(w)
           endif
       endif
   endfor
```

Ilustrasi:



- DFS(1): v=1; dikunjungi[1]=true; DFS(2)
  - DFS(2): v=2; dikunjungi[2]=true; DFS(4)
    - DFS(4): v=4; dikunjungi[4]=true; DFS(8)
      - DFS(8): v=8; dikunjungi[8]=true; DFS(5)
        - DFS(5): v=5; dikunjungi[5]=true
        - DFS(6): v=6; dikunjungi[6]=true; DFS(3)
          - DFS(3): v=3; dikunjungi[3]=true; DFS(7)
            - DFS(7): v=7; dikunjungi[7]=true

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

2.2 C# Desktop Application Development

C# Desktop Application Development adalah sebuah kakas yang yang berbasis desktop yang mampu beroperasi tanpa menggunakan internet. Aplikasi berbasis desktop biasanya dibuat menggunakan 3 bahasa pemrograman, yaitu C++, C#, dan Java. C# sendiri dapat dibantu dengan framework .Net yang memudahkan programmer dalam melakukan pengkodingan karena bentuk visual aplikasinya yang sudah langsung terlihat oleh programmer-nya sehingga proses debugging menjadi lebih mudah. Keunggulan desktop application yang menggunakan framework .Net dan bahasa pemrograman C# adalah developer dimungkinkan untuk membuat aplikasi windows base Laporan Tugas Besar II IF2211 Strategi Algoritma 5 yang di-launch melalui build di dalam Visual Studio. Keunggulan lainnya adalah dapat berjalan independen, tanpa perlu menggunakan browser, tidak perlu koneksi internet, karena

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah Pemecahan Masalah

Langkah pemecahan masalah pada kasus ini terbagi menjadi 2, yaitu penyelesaian secara DFS dan penyelesaian secara BFS dengan menggunakan graf dinamis. Graf dinamis digunakan karena dalam masalah pencarian ini, ketika file yang dicari sudah ditemukan, maka pencarian tidak perlu dilanjutkan lagi sehingga tidak pembangkitan simpul lain sangat bergantung pada kondisi penemuan file yang dicari. Langkah yang dilakukan pertama, *Starting Directory* yang telah dipilih akan dianggap sebagai simpul akar dari sebuah graf. Dari simpul akar graf tersebut, akan dilakukan penelusuran terhadap semua anaknya (membangkitkan simpul anak). Anak dalam hal ini adalah semua folder (subfolders) dan files yang ada di dalam simpul akar graf tersebut. Pembangkitan dilakukan dengan menggunakan program untuk mendapatkan semua nama files yang ada pada folder/akar graf dan semua subfolders yang ada. Files akan dianggap sebagai simpul daun (simpul yang tidak memiliki anak) dan akan dilakukan pengecekan terlebih dahulu dibanding dengan folder karena pengecekan pada simpul daun terlebih dahulu dapat memiliki kemungkinan menemukan file yang dicari lebih cepat sehingga nantinya tidak perlu melakukan pembangkitan simpul yang lain. Jika tidak ada files yang cocok, maka pencarian akan dilakukan dengan melakukan penelusuran pada folder yang menjadi anak dari simpul akar. Dalam kasus ini, folder yang menjadi anak dari simpul akar dapat dikatakan sebagai upa-graf sehingga penelusuran dapat dilakukan dengan cara yang sama seperti sebelumnya hingga file yang dicari ditemukan atau semua simpul (files dan folders) telah dikunjungi. Jika pencarian mensyaratkan untuk memunculkan semua kemungkinan files yang dicari, maka pencarian akan dilakukan hingga semua simpul (files dan folders) telah dikunjungi sembari menyimpan *path directory* file yang telah ditemukan pada sebuah list.

Pemecahan masalah ini juga akan memanfaatkan beberapa *standard library* yang ada pada bahasa C#, yaitu list, queue, dan stack. List digunakan untuk menyimpan *path directory* file yang telah ditemukan. Queue digunakan dalam proses BFS dan stack digunakan dalam proses DFS. Queue dan Stack digunakan untuk menyimpan simpul yang telah dibangkitkan namun belum dikunjungi (setiap pembangkitan simpul, simpul akan dimasukkan ke dalam queue ataupun stack).

3.2 Proses Mapping Persoalan

Penyelesaian persoalan ini terbagi menjadi dua, yaitu penyelesaian dengan cara DFS dan dengan cara BFS. Untuk DFS, langkah-langkah yang dilakukan untuk penyelesaian persoalan adalah sebagai berikut:

1. Program akan membuat sebuah list untuk menyimpan hasil dan stack untuk menyimpan simpul.
2. Program menerima *Starting Directory* dan akan menjadikannya sebagai simpul akar dari graf yang akan dibentuk secara dinamis.
3. Dari simpul akar atau *Starting Directory*, kemudian bangkitkan simpul-simpul anaknya dengan memasukkannya pada Stack yang telah dibuat. Pembangkitan simpul dilakukan dengan menggunakan program untuk memperoleh nama folder dan file yang ada pada simpul akar.
4. Kemudian, lakukan penelusuran pada tiap simpul yang ada pada Stack. Penelusuran dilakukan dengan mengakses simpul yang berada pada urutan pertama Stack dan kemudian membuang simpul tersebut dari Stack (melakukan pop pada Stack).
5. Jika simpul ternyata adalah sebuah file, maka lakukan pencocokan dengan file yang dicari. Jika cocok, maka masukkan *directory path*-nya ke dalam list yang telah dibuat.
6. Jika simpul ternyata adalah sebuah folder, maka lakukan pembangkitan anaknya (files dan subfolders yang ada pada folder tersebut) dan masukkan ke dalam Stack.
7. Langkah 4, 5, dan 6 terus dilakukan jika disyaratkan untuk menampilkan seluruh kemungkinan files yang ditemukan atau jika file belum ditemukan. Jika tidak disyaratkan untuk menampilkan seluruh kemungkinan file dan file telah ditemukan, maka program dapat dihentikan pada langkah ke 5.

Untuk BFS, langkah yang digunakan juga hampir sama, yaitu sebagai berikut:

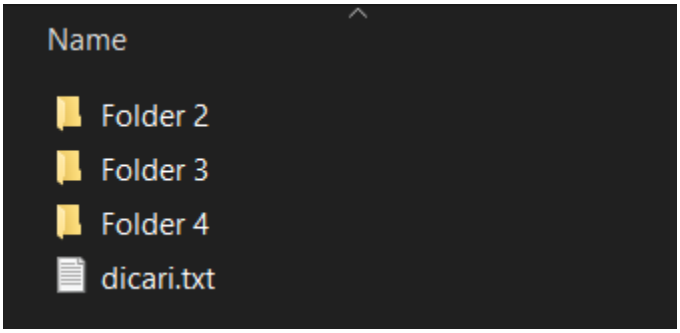
1. Program akan membuat sebuah list untuk menyimpan hasil dan queue untuk menyimpan simpul.
2. Program menerima *Starting Directory* dan akan menjadikannya sebagai simpul akar dari graf yang akan dibentuk secara dinamis.
3. Dari simpul akar atau *Starting Directory*, kemudian bangkitkan simpul-simpul anaknya dengan memasukkannya pada Queue yang telah dibuat. Pembangkitan simpul dilakukan dengan menggunakan program untuk memperoleh nama folder dan file yang ada pada simpul akar.
4. Kemudian, lakukan penelusuran pada tiap simpul yang ada pada Queue. Penelusuran dilakukan dengan mengakses simpul yang berada pada urutan pertama Queue dan kemudian membuang simpul tersebut dari Queue (melakukan dequeue pada Queue).
8. Jika simpul ternyata adalah sebuah file, maka lakukan pencocokan dengan file yang dicari. Jika cocok, maka masukkan *directory path*-nya ke dalam list yang telah dibuat.
9. Jika simpul ternyata adalah sebuah folder, maka lakukan pembangkitan anaknya (files dan subfolders yang ada pada folder tersebut) dan masukkan ke dalam Queue.

10. Langkah 4, 5, dan 6 terus dilakukan jika disyaratkan untuk menampilkan seluruh kemungkinan files yang ditemukan atau jika file belum ditemukan. Jika tidak disyaratkan untuk menampilkan seluruh kemungkinan file dan file telah ditemukan, maka program dapat dihentikan pada langkah ke 5.

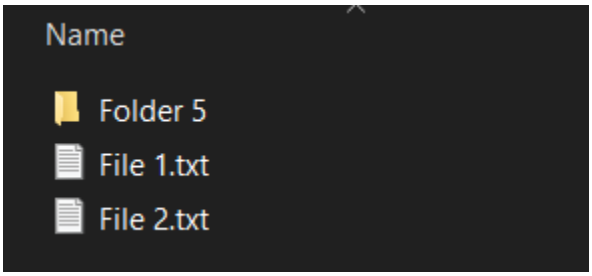
3.3 Contoh Ilustrasi Kasus Lain

Contoh lain pada kasus Folder Crawling ini adalah sebagai berikut:

Starting Directory pada kasus ini adalah Folder 1 dan akan dilakukan pencarian pada file dicari.txt dengan opsi *Find All Occurance*. Adapun isi dari Folder 1 berupa

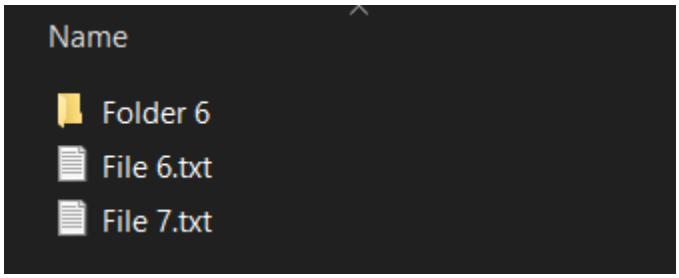


Pada Folder 2 berisi



Pada Folder 3 berisi

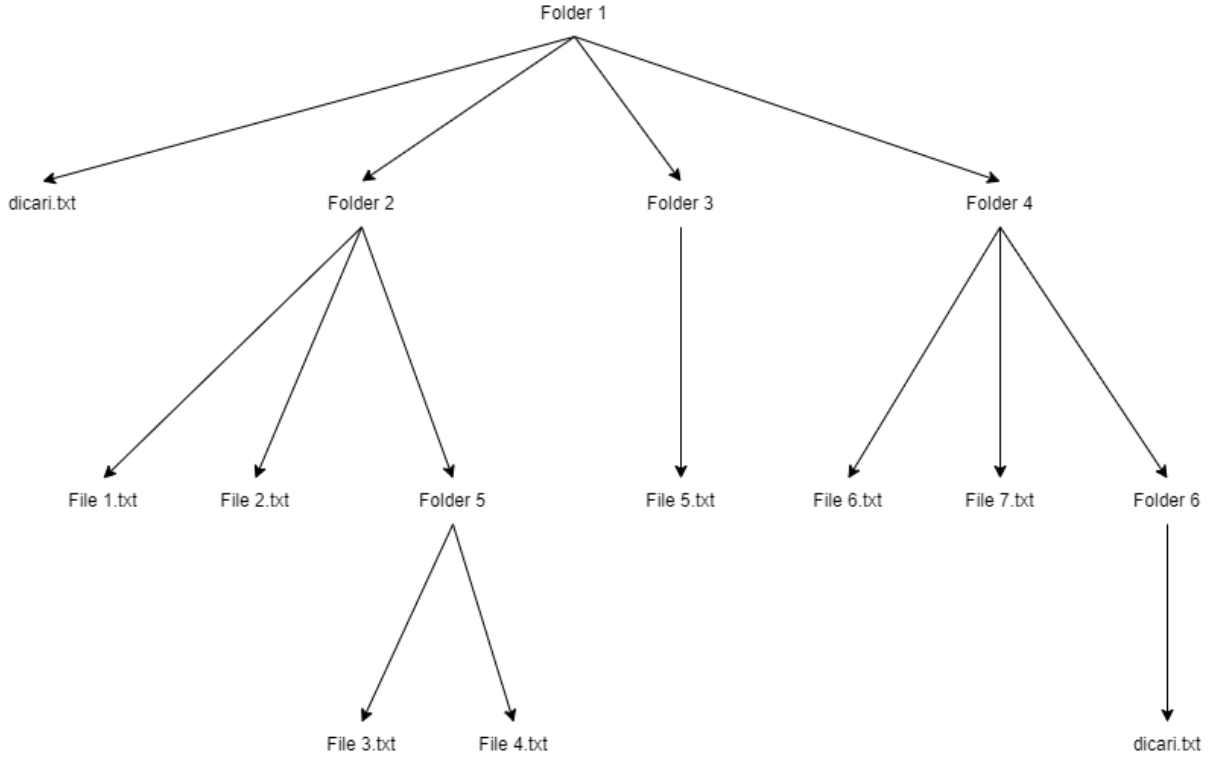
Pada Folder 4 berisi



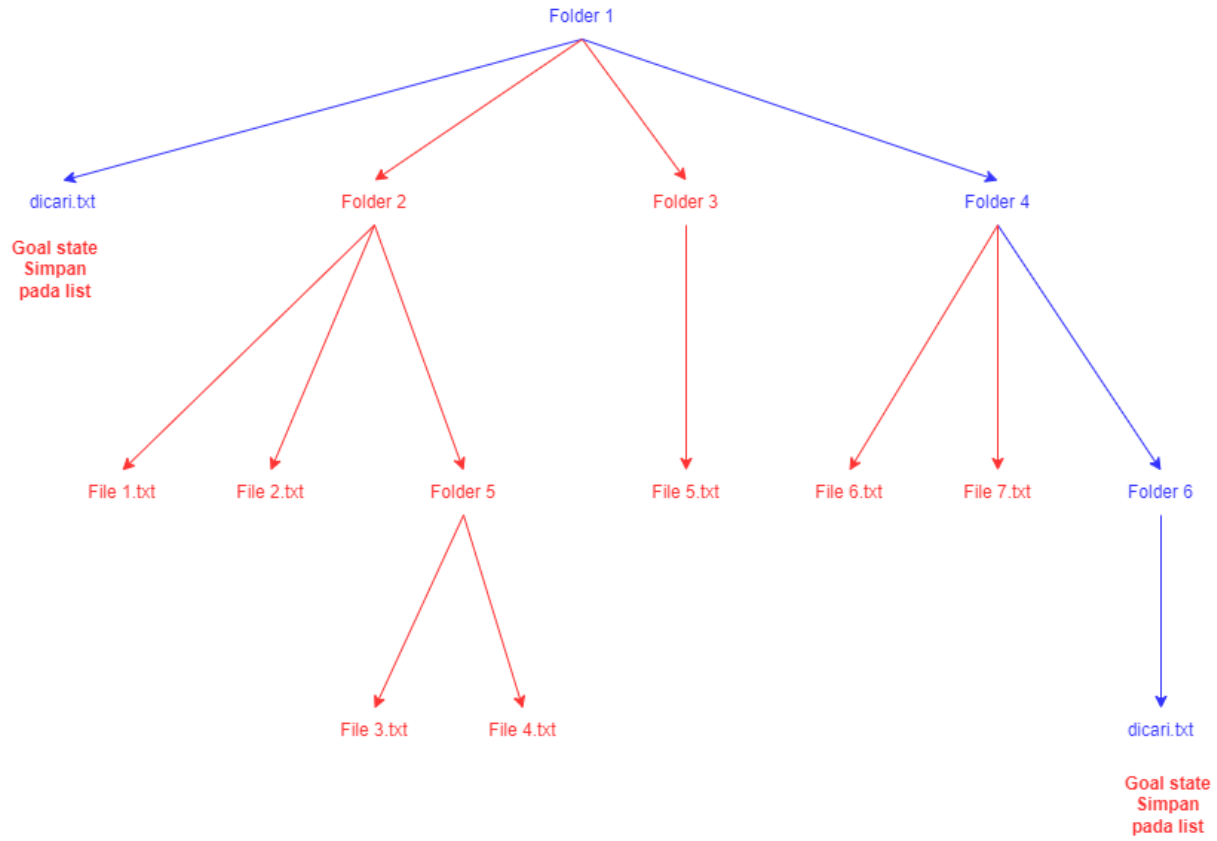
Pada Folder 5 berisi

Pada Folder 6 berisi

Atau jika dibuat menjadi tree akan terlihat sebagai berikut:

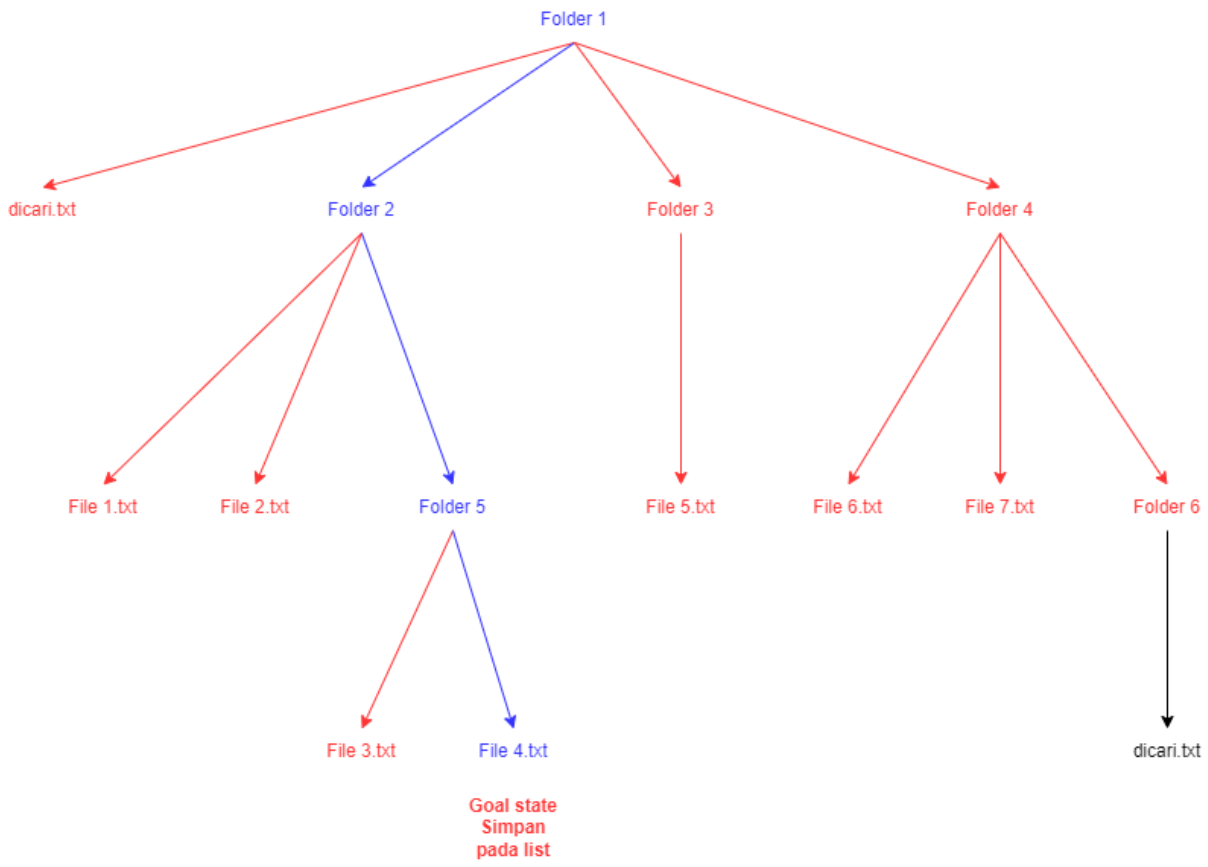


Maka dengan pencarian secara BFS, pencarian akan berjalan sebagai berikut:

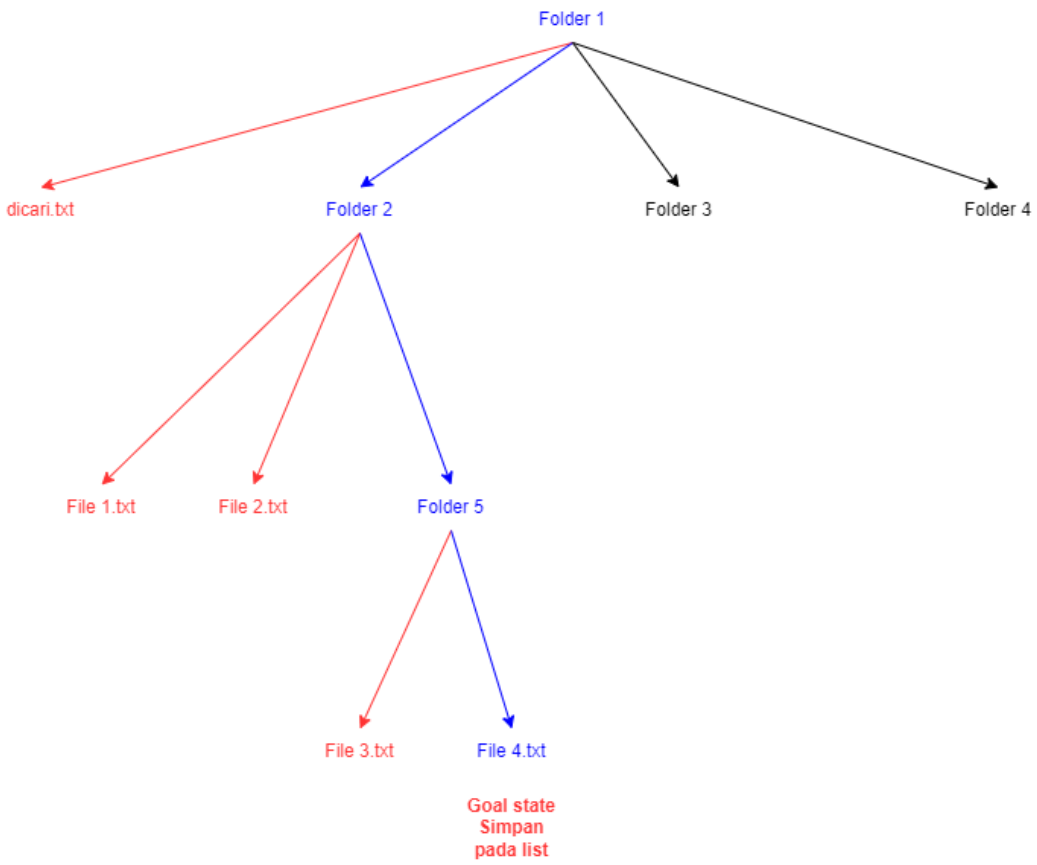


Dengan pencarian secara DFS, tree yang dihasilkan juga akan sama karena disyaratkan untuk memunculkan semua kemungkinan file yang ditemukan sehingga perlu melakukan pengecekan pada semua subgraf yang ada.

Jika kasus diubah untuk mencari File 4.txt, maka hasil dari pencarian dengan BFS akan sebagai berikut:



Dan jika menggunakan DFS maka hasilnya akan sebagai berikut:



BAB IV
IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Program

Pseudocode:

```
Procedure mainProgramGUI()
      path, filename : string
      isFindAll, isBFS, isDFS : boolean
      path = StartingDirectoryBrowseButton()
      filename = FileNameTextField()
      isFindAll = FindAllCheckBox() //true if checked, false if not checked
      isBFS = BFSRadioButton()     //true if clicked
      isDFS = DFSRadioButton()    //true if clicked
      if (SearchButton.clicked) then
            if (isBFS) then
                  SearchBFS(filename, path, isFindAll)
            else
                  SearchDFS(filename, path, isFindAll)
      OutputTextField()
      ResultTextField()
```

```
Function StartingDirectoryBrowseButton() -> string
      if (clicked) then
            return ExploreFolder()  //function to explore folder and choose
            one of them
```

```
Function FileNameTextFiled() -> string
      return getText()   //function to get text from keyboard input
```

```
//global variable
results : List<string>
fs_path : List<string>
nt_path : List<string>
f_node : new Node("root", null);

Procedure searchBFS(search_file : string, path : string, isFindAll :
boolean)
paths : Queue<string>
isFound : boolean
isFound = false
newpath : string
paths.Enqueue(path)
tmp : Node
while ((!paths.isEmpty()) && (!isFound)) do
```

```
      newpath = paths.Dequeue()
      tmp = f_node.AddFolderNode(newpath)
      tmp.tr = true
      fs_path.Add(newpath)
      DirectoryInfo dir = new DirectoryInfo(newpath);
      FileInfo[] files = dir.GetFiles();
      for each (FileInfo file in files)   //get all files from newpath
            if (!isFound) then
                  fs_path.Add(file.FullName);
                  tmp = f_node.AddFolderNode(file.FullName);
                  if (file.Name == search_file)
                        results.Add(file.FullName);
                        tmp.SetFound();
                        if (!isFindAll) then
                              isFound = true;
      if (!isFound) then  //Add all subfolder to Queue
            string[] folders = Directory.GetDirectories(newpath);
            for each (var folder in folders)
                  f_node.AddFolderNode(folder).tr = false;
                  paths.Enqueue(folder);
// Hilangin Root Path
fs_path.RemoveAt(0);
// Yang di queue tapi belum di-traverse
nt_path = paths.ToList();
```

```
//global variable
results : List<string>
fs_path : List<string>
nt_path : List<string>
f_node : new Node("root", null);

Procedure searchDFS(search_file : string, path : string, isFindAll :
boolean)
paths : Stack<string>
paths.Push(path);
isFound : boolean
newpath : string
isFound = false
Node tmp
while ((!paths.isEmpty()) && (!isFound)) do
      newpath = paths.Pop()
      tmp = f_node.AddFolderNode(newpath)
      tmp.tr = true
      fs_path.Add(newpath)
      DirectoryInfo dir = new DirectoryInfo(newpath)
      FileInfo[] files = dir.GetFiles();
      for each (FileInfo file in files)
            if (!isFound) then
```

```
                  fs_path.Add(file.FullName)
                  tmp = f_node.AddFolderNode(file.FullName)
                  if (file.Name == search_file) then
                        results.Add(file.FullName)
                        tmp.SetFound()
                        if (!isFindAll) then
                              isFound = true
      if (!isFound) then
            //Add subfolder from current folder
            string[] folders = Directory.GetDirectories(newpath)
            for each (var folder in folders.Reverse())
                  f_node.AddFolderNode(folder).tr = false
                  paths.Push(folder)
// Hilangin Root Path
fs_path.RemoveAt(0)
// Yang di stack tapi belum di-traverse
nt_path = paths.ToList()
```

```
Procedure OutputTextField()
      if (search-process-end) then
            ShowGraphVisualizer()   //Show searching process in graph
```

```
Procedure ResultTextField()
      if (search-process-end) then
            ShowResultFound()   //Show result from searching process
```

B. Penjelasan Struktur Data dan Spesifikasi Program
Yang Highlight Kuning masih bingung takut salah

1. Class FolderCrawler
   Class FolderCrawler merupakan Class yang berisikan Algoritma BFS dan DFS untuk digunakan di program ini. Class ini hanya terdiri dari dua method yaitu, method BFS dan DFS. Pada method BFS, method ini menggunakan konsep queue untuk menyimpan simpul yang telah dibangkitkan akan tetapi belum dikunjungi. Sedangkan pada DFS, method ini menggunakan Stack untuk menyimpan simpul yang telah dibangkitkan namun belum dikunjungi. Class ini berguna untuk mengimplementasikan Algoritma dari BFS dan DFS yang akan dibuat untuk mencari rute atau jalur path file yang ingin dicari dari suatu folder yang nantinya akan dibuat sebuah graf sebagai visualisasinya.

2. Class GVisualizer
   Class ini merupakan Class yang digunakan untuk memvisualisasikan graf yang dibentuk dari proses pencarian menggunakan algoritma BFS atau DFS pada Class Folder Crawling. Class ini terdapat beberapa method yaitu, GVisualizer untuk

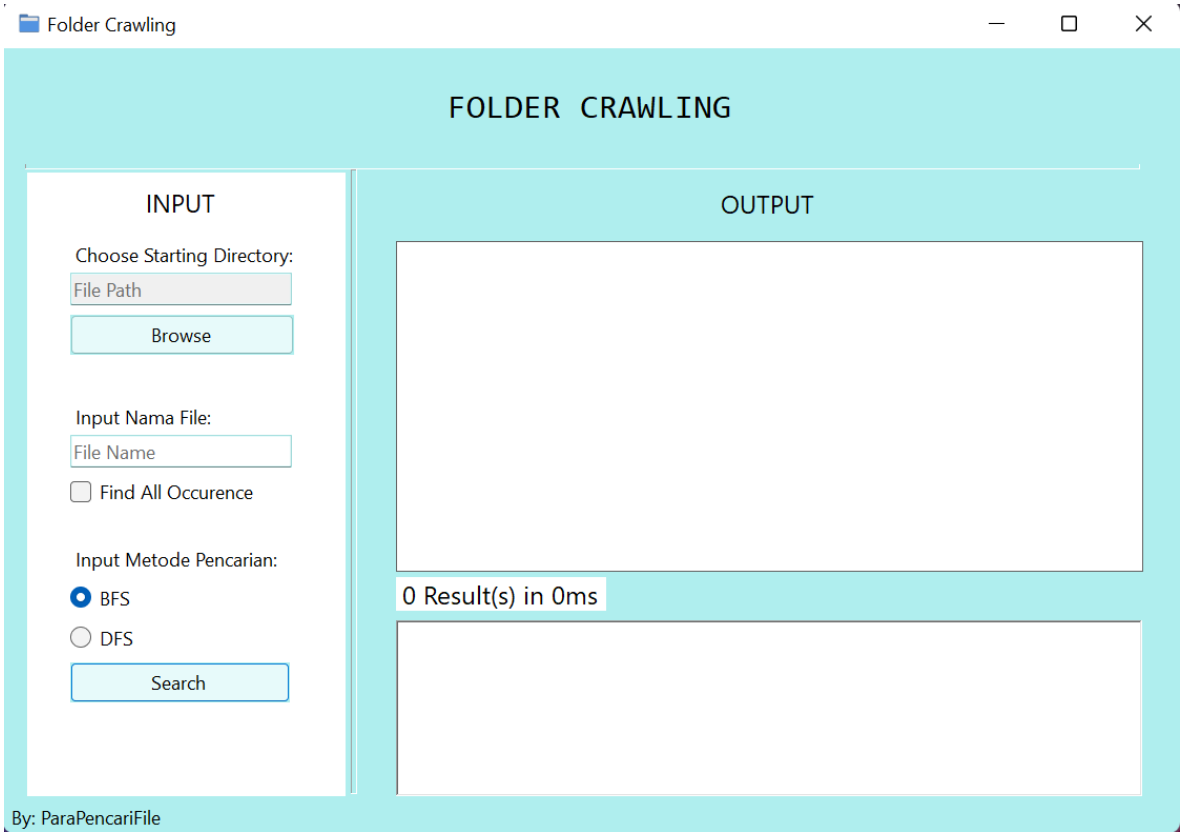
menambahkan node berupa root pada graf, FolderParent untuk mendapatkan folder parent yang diinginkan, FolderName untuk mendapatkan nama dari folder, Parse memastikan kondisi Node bisa di-parse lalu memanggil RecursiveParse , RecursiveParse melakukan parsing Node secara rekursif, dan Show untuk menampilkan hasil dari graf yang terbentuk

3. Class Node
   Class Node ini merupakan Class yang digunakan untuk mengimplementasikan struktur data tree untuk membuat percabangan solusi simpul dan garis yang nanti dikombinasikan dengan algoritma BFS dan DFS pada class FolderCrawler. Class ini memiliki beberapa method seperti, FolderParent untuk mendapatkan folder utama, FolderName untuk mendapatkan nama dari foldernya, ParseFolderName untuk menguraikan nama-nama folder dari file ke root path, AddFolderNode untuk menambahkan node pada folder, AddChildern untuk menambahkan children pada parent. getParent untuk mendapatkan parent, GetName untuk mendapatkan nama dari folder, getChild untuk mendapatkan child dari parent, setFound untuk mengecek apakah ketemu atau tidak, toString() dan recToString() mengubah meng-output Node sebagai string, dan toRoot() traverse dari node manapun ke Node Root.

4. Class Program
   Class Program ini merupakan Class utama dari program ini yang berguna untuk menginisialisasi konfigurasi aplikasi dan juga menjalankan aplikasi dari program yang telah dibuat

5. Form1
   Class Form1 ini merupakan Class yang berisikan kode kode yang terintegrasi langsung dengan aplikasi, seperti button1_click yang apabila di klik maka langsung menjalankan program algoritma dan visualizer. lalu , button2_click yang apabila di klik maka berguna untuk membuka folder untuk mencari folder yang diinginkan. Lalu, terdapat richTextBox1_LinkClicked yang berguna untuk menghasilkan output berupa path dari file yang dicari dalam suatu folder tertentu dan apabila di klik maka kita akan langsung ke file tersebut.

C. Tata Cara Penggunaan Program
   Untuk menggunakan program dari kelompok kami, para user atau pengguna dapat menjalankan program yang bernama <nama-file>.exe dan program akan berjalan normal. Kemudian program akan menampilkan interface yang berisi beberapa fitur yang dapat dilihat dibawah ini.



Gambar C.1 Tampilan Awal Program

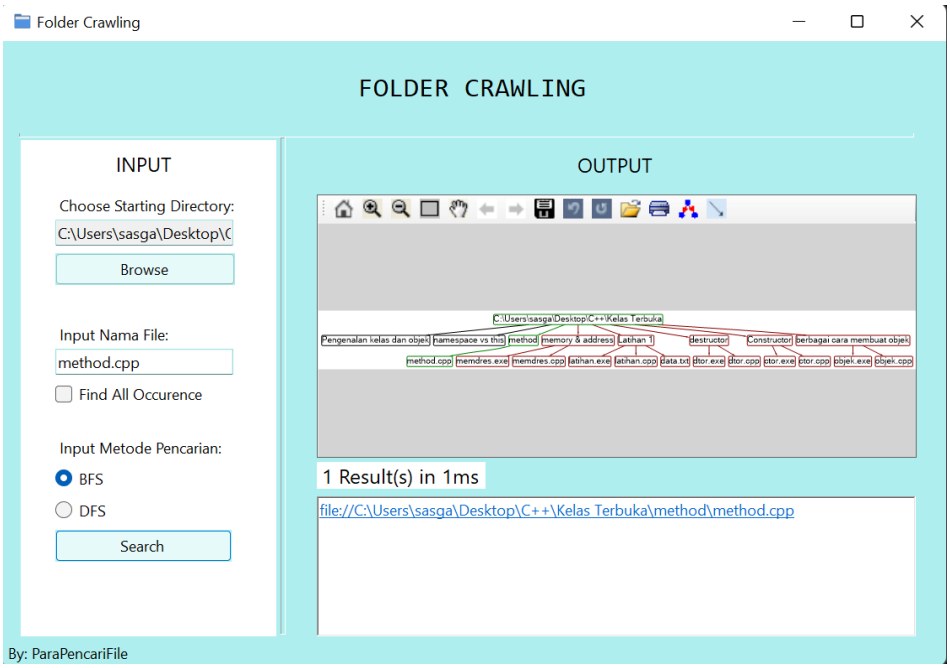
Dapat dilihat bahwa program diatas akan memberikan fitur kepada pengguna untuk melakukan browse *path* sesuai keinginan pengguna. Lalu, pengguna dapat menuliskan nama file yang ingin dicari untuk semua tipe file (pdf,txt, dan lain lain) dan dapat memilih apakah ingin dicari semua kemungkinan atau tidak. Setelah itu, pengguna dapat memilih salah satu algoritma apa yang ingin dipakai, BFS atau DFS, dan klik tombol search.

Setelah tombol search di klik, maka program akan memvisualisasikan sebuah graf yang terdiri dari simpul dan sisi yang berwarna yang menunjukkan jalur solusi dari tiap masing-masing solusi, hijau untuk solusi yang ditemukan, merah untuk solusi yang tidak ditemukan, dan hitam, simpul yang belum dicek. Selain itu juga, program ini memberikan fitur berupa informasi waktu yang diperlukan untuk mencari suatu file dan terdapat berapa file yang ditemukan dengan nama file yang di input oleh pengguna. Terakhir, apabila file ditemukan, maka program akan memberikan sebuah path file dimana lokasi file yang dicari ditemukan, apabila file tidak ditemukan maka tidak akan menghasilkan apa apa.

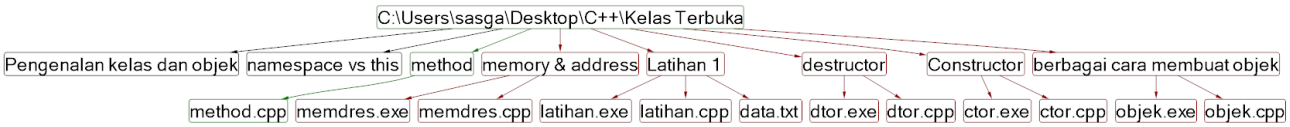
D. Hasil Pengujian

a. Data Uji 1 - BFS

Data uji yang pertama diambil dari folder “Kelas Terbuka” dengan mencari nama file “method.cpp” seperti tampilan dibawah. Dapat dilihat juga bahwa program ini menghasilkan visualisasi dari pencarian file tersebut menggunakan graf dengan menggunakan algoritma BFS. Simpul dan Garis yang berwarna hijau merupakan solusi untuk pencarian file “method.cpp”.

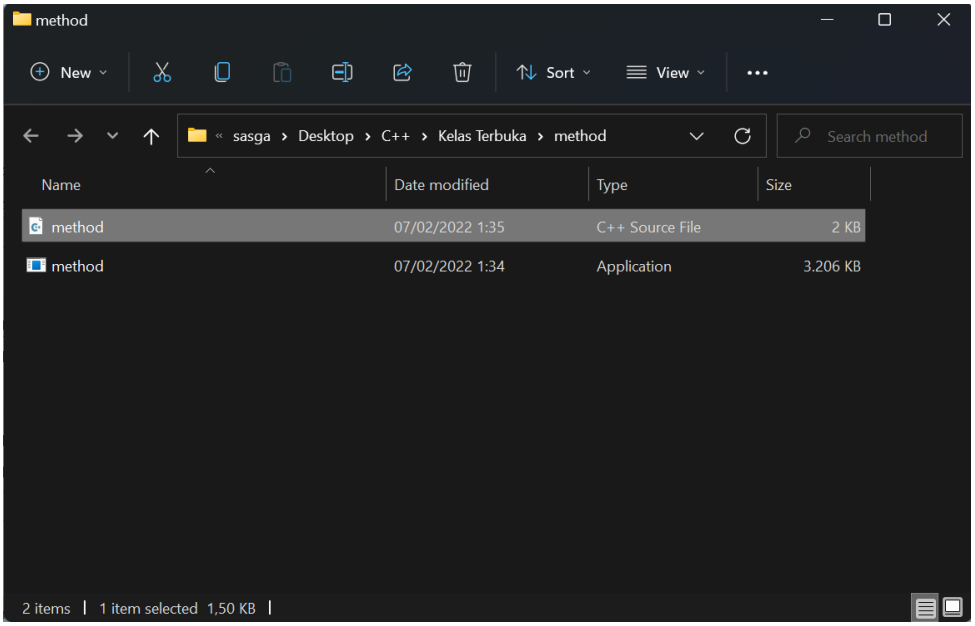


Gambar D.a.1 Hasil Pencarian File “Method.cpp”



Gambar D.a.2 Simpul dan Garis yang terbentuk

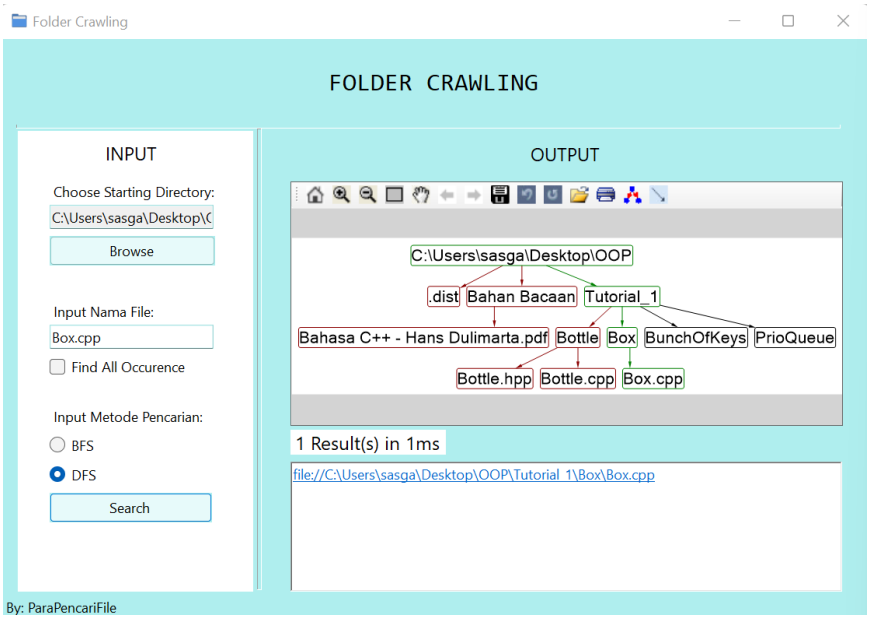
Lalu, apabila pengguna ingin mengetahui letak file tersebut berada, pengguna bisa mengklik link path file yang diberikan program dan program akan langsung menampilkannya seperti gambar berikut:



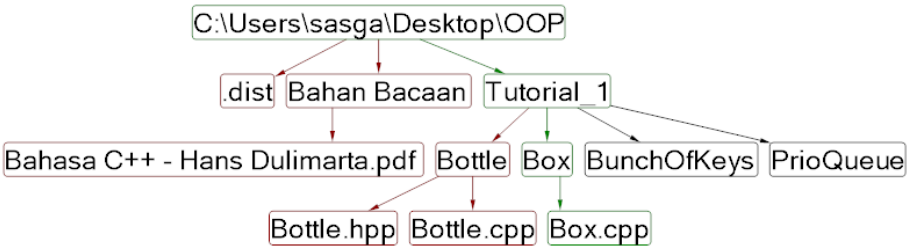
Gambar D.a.3 Lokasi File yang Dicari

b. Data Uji 2 - DFS

Data uji yang kedua diambil dari folder "OOP" dengan mencari nama file "Box.cpp" seperti tampilan dibawah. Dapat dilihat juga bahwa program ini menghasilkan visualisasi dari pencarian file tersebut menggunakan graf dengan menggunakan algoritma DFS. Simpul dan Garis yang berwarna hijau merupakan solusi untuk pencarian file "Box.cpp".

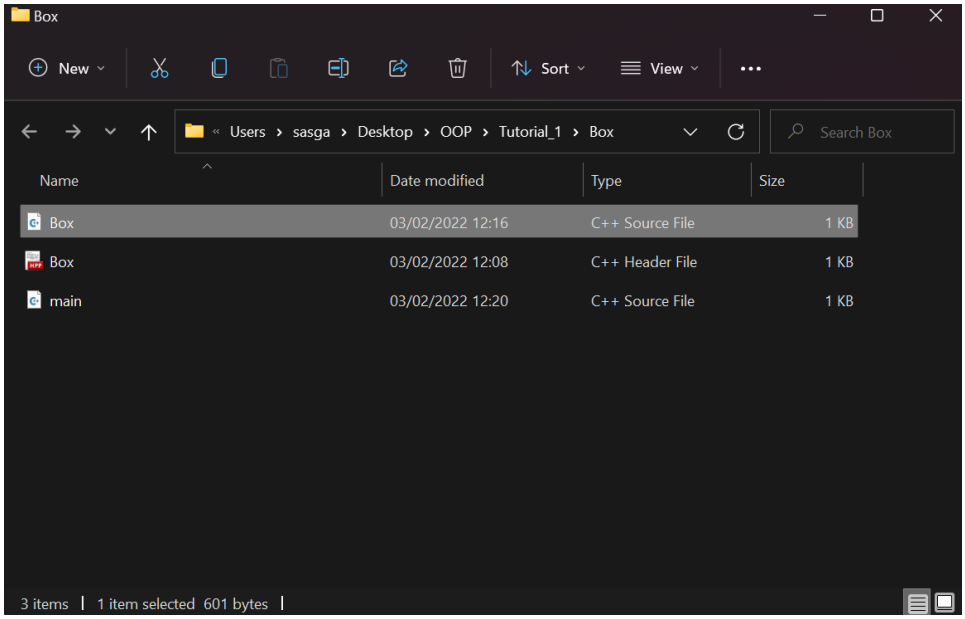


Gambar D.b.1 Hasil Pencarian File "Box.cpp"



Gambar D.b.2 Simpul dan Garis yang terbentuk

Lalu, apabila pengguna ingin mengetahui letak file tersebut berada, pengguna bisa mengklik link path file yang diberikan program dan program akan langsung menampilkannya seperti gambar berikut:

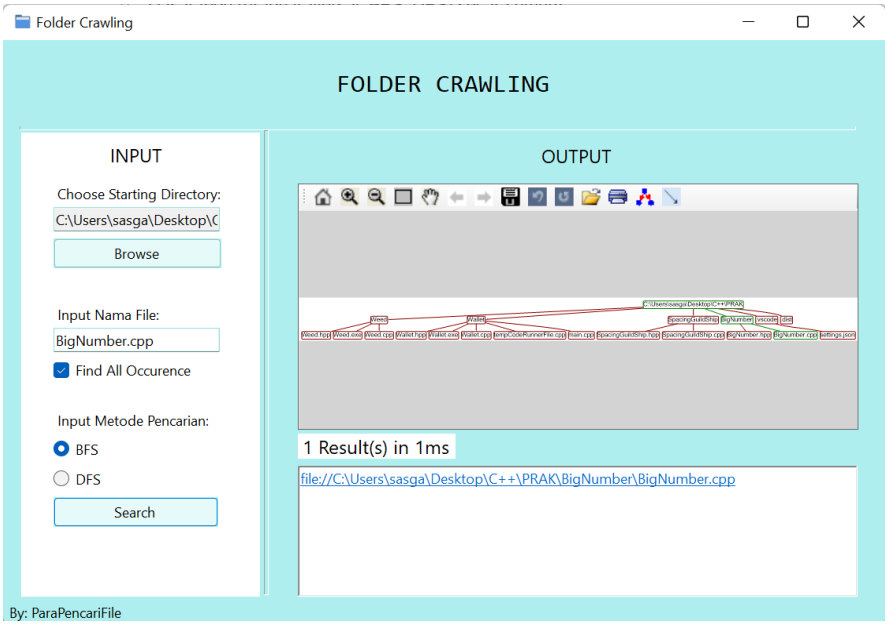


Gambar D.b.3 Lokasi File yang Dicari

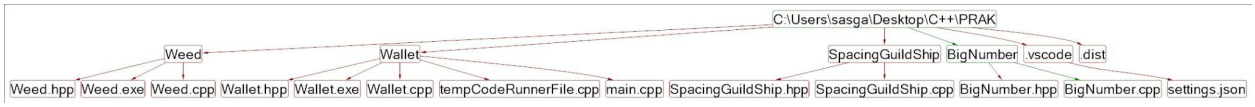
c. Uji Coba 3 - All Occurence (BFS)

Data uji yang kedua diambil dari folder “PRAK” dengan mencari nama file “BigNumber.cpp” seperti tampilan di bawah. Dapat dilihat juga bahwa program ini menghasilkan visualisasi dari pencarian file tersebut menggunakan graf dengan menggunakan algoritma BFS dan mencari semua kemungkinan yang dapat terjadi.

Simpul dan Garis yang berwarna hijau merupakan solusi untuk pencarian file “BigNumber.cpp”.

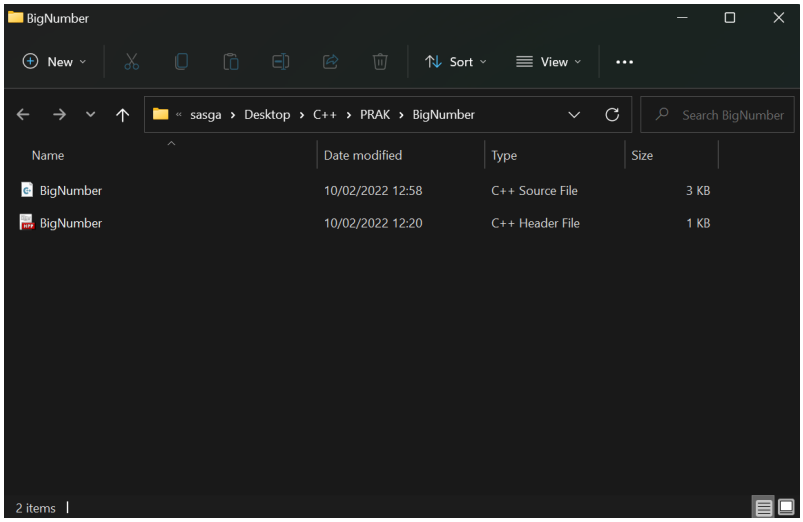


Gambar D.c.1 Hasil Pencarian File “BigNumber.cpp”



Gambar D.c.2 Simpul dan Garis yang terbentuk

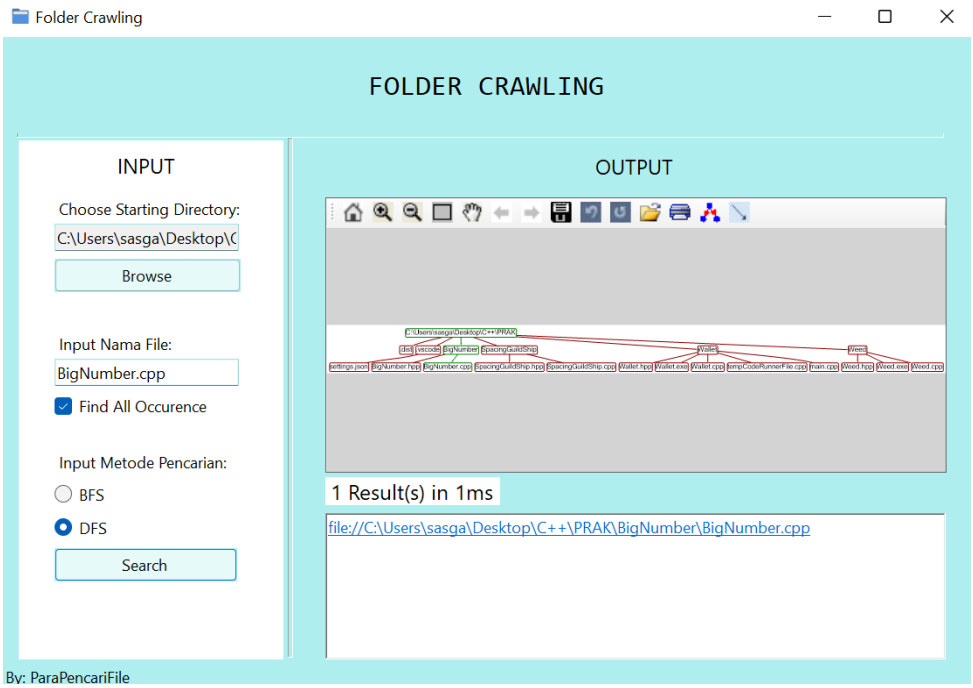
Lalu, apabila pengguna ingin mengetahui letak file tersebut berada, pengguna bisa mengklik link path file yang diberikan program dan program akan langsung menampilkannya seperti gambar berikut:



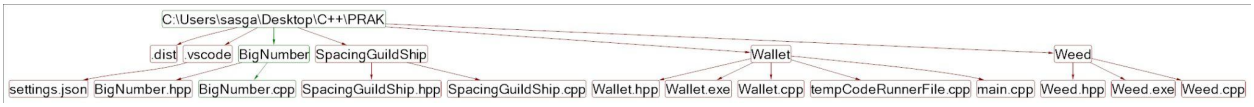
Gambar D.b.3 Lokasi File yang Dicari

d. Uji Coba 4 - All Occurence (DFS)

Data uji yang kedua diambil dari folder "PRAK" dengan mencari nama file "BigNumber.cpp" seperti tampilan dibawah. Dapat dilihat juga bahwa program ini menghasilkan visualisasi dari pencarian file tersebut menggunakan graf dengan menggunakan algoritma DFS dan mencari semua kemungkinan yang dapat terjadi. Simpul dan Garis yang berwarna hijau merupakan solusi untuk pencarian file "BigNumber.cpp".

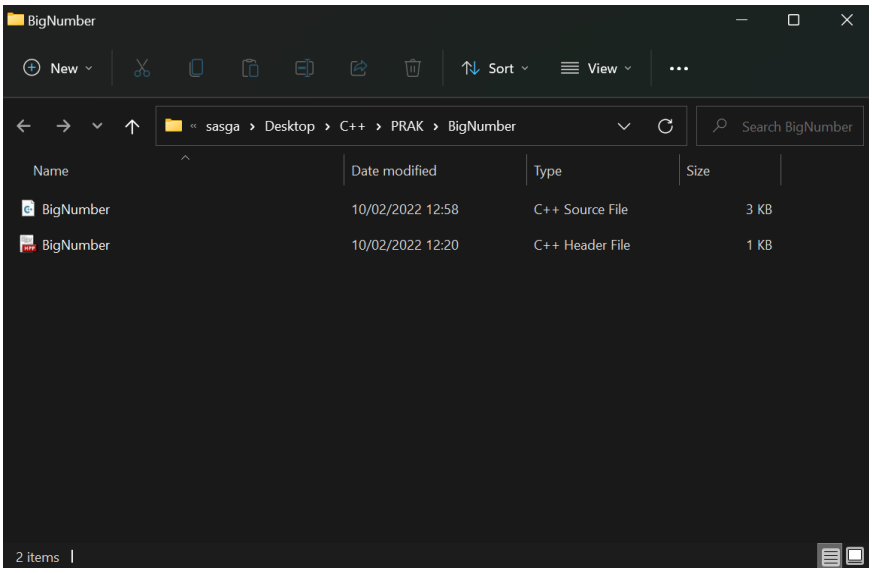


Gambar D.d.1 Hasil Pencarian File "BigNumber.cpp"



Gambar D.d.2 Simpul dan Garis yang terbentuk

Lalu, apabila pengguna ingin mengetahui letak file tersebut berada, pengguna bisa mengklik link path file yang diberikan program dan program akan langsung menampilkannya seperti gambar berikut:



Gambar D.b.3 Lokasi File yang Dicari

E. Analisis dari Desain Solusi Algoritma BFS dan DFS yang Diimplementasikan

Berdasarkan desain solusi algoritma BFS dan DFS yang diimplementasikan, dapat dilihat bahwa dari segi waktu, kecepatan algoritma DFS maupun BFS sangat bergantung pada letak dicarinya file. Dilihat dari test case yang diberikan, apabila simpul tujuan (simpul yang dicari) pada representasi graf memiliki kedalaman yang relatif tinggi terhadap simpul akar maka dilihat dari sudut pandang kompleksitas, algoritma BFS akan cenderung lebih besar daripada algoritma DFS. Sedangkan, apabila simpul tujuan (simpul yang dicari) pada representasi graf memiliki kedalaman yang relatif rendah terhadap simpul akar maka kompleksitas dari algoritma DFS akan cenderung lebih besar daripada algoritma BFS. Hal tersebut menunjukkan bahwa algoritma DFS lebih cepat dalam persoalan yang memiliki kedalaman yang tinggi. Sedangkan algoritma BFS lebih cepat dalam persoalan yang berjarak dekat dari simpul akar.

BAB V
KESIMPULAN, SARAN, DAN REFLEKSI

1. Kesimpulan

Berikut adalah beberapa kesimpulan yang kami dapatkan dari pengerjaan tugas besar Mata Kuliah IF2211 Strategi Algoritma

a. BFS dan DFS adalah salah satu metode untuk melakukan penjelajahan simpul dari representasi graf. BFS untuk melakukan penjelajahan secara menyamping (*breadth*), dan DFS untuk melakukan penjelajahan secara mendalam (*depth*).
b. Pencarian File dalam suatu folder seperti di File Explorer dapat ditelusuri dengan menggunakan implementasi dari algoritma BFS dan DFS.
c. Kelompok kami berhasil dalam membuat aplikasi berbasis desktop untuk mencari File dalam suatu folder tertentu yang direpresentasikan ke dalam graf.
d. *Interface* aplikasi dari kelompok kami dibuat dengan *framework* .NET dengan menggunakan bahasa pemrograman C#.
e. Program yang kelompok kami telah buat, berhasil untuk menemukan solusi-solusi yang menggunakan algoritma BFS dan DFS dengan baik dan benar.

2. Saran

Untuk waktu kedepan, diharapkan bagi kami untuk mempelajari terlebih dahulu bagaimana menggunakan visualisasi graf dan cara membuat aplikasi desktop yang baik sehingga waktu pengerjaan dalam membuat aplikasi berbasis desktop ini dapat dipangkas.

Link Github : https://github.com/haidarihza/Tubes2-Stima.git

Link Video Demo : https://youtu.be/U227fW7QvRc

Daftar Pustaka

https://docs.microsoft.com/en-us/dotnet/csharp/

http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf