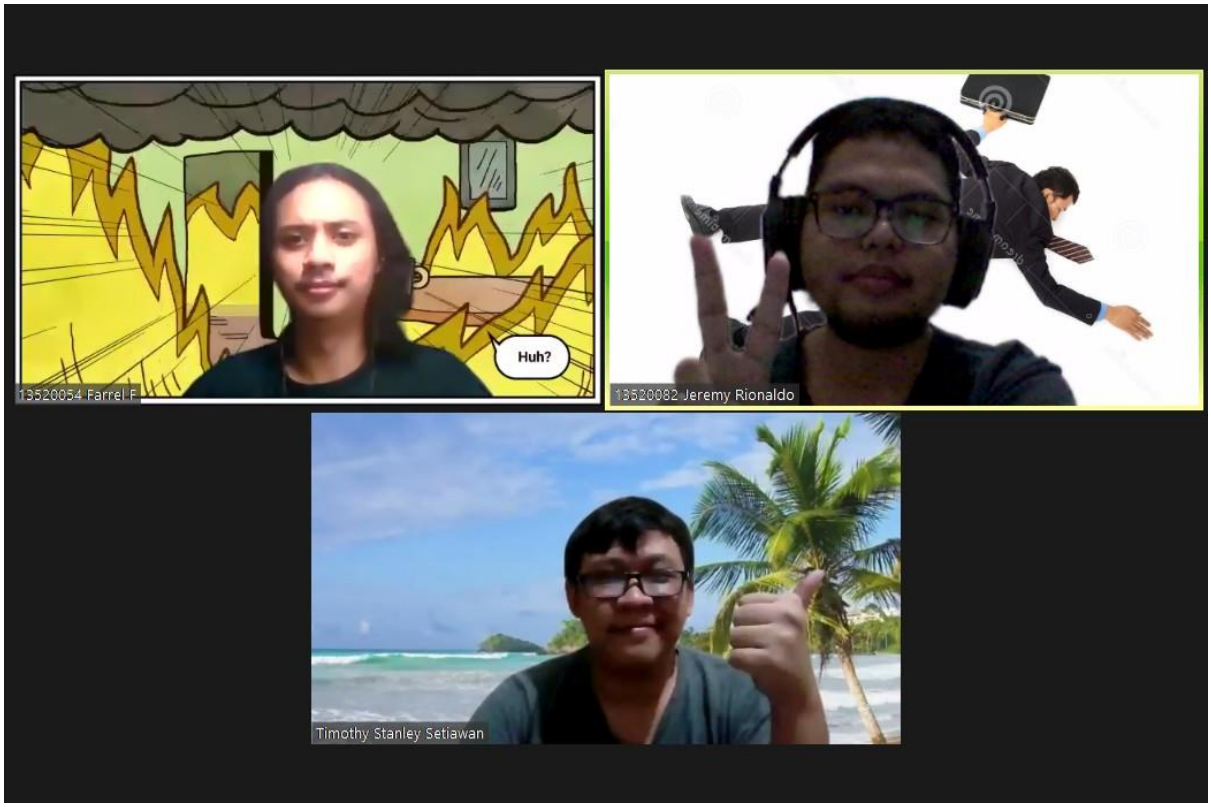
Laporan Tugas Besar 1 IF2211 Strategi Algoritma
Semester II tahun 2021/2022

Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder Crawling*



Dipersiapkan oleh:

Kelompok 41: Barrel Bastian Firat Search

13520028	Timothy Stanley Setiawan
13520054	Farrel Farandieka Fibriyanto
13520082	Jeremy Rionaldo Pasaribu

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

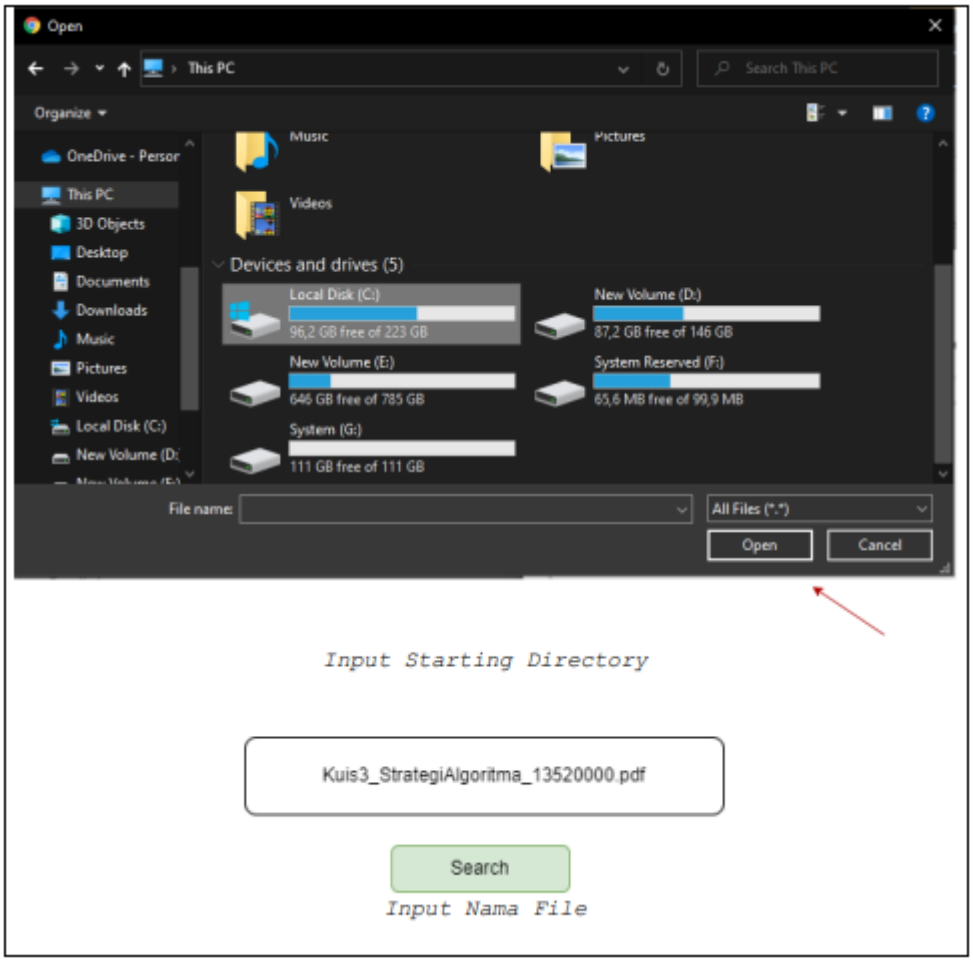
Daftar Isi

BAB 1: Deskripsi Masalah

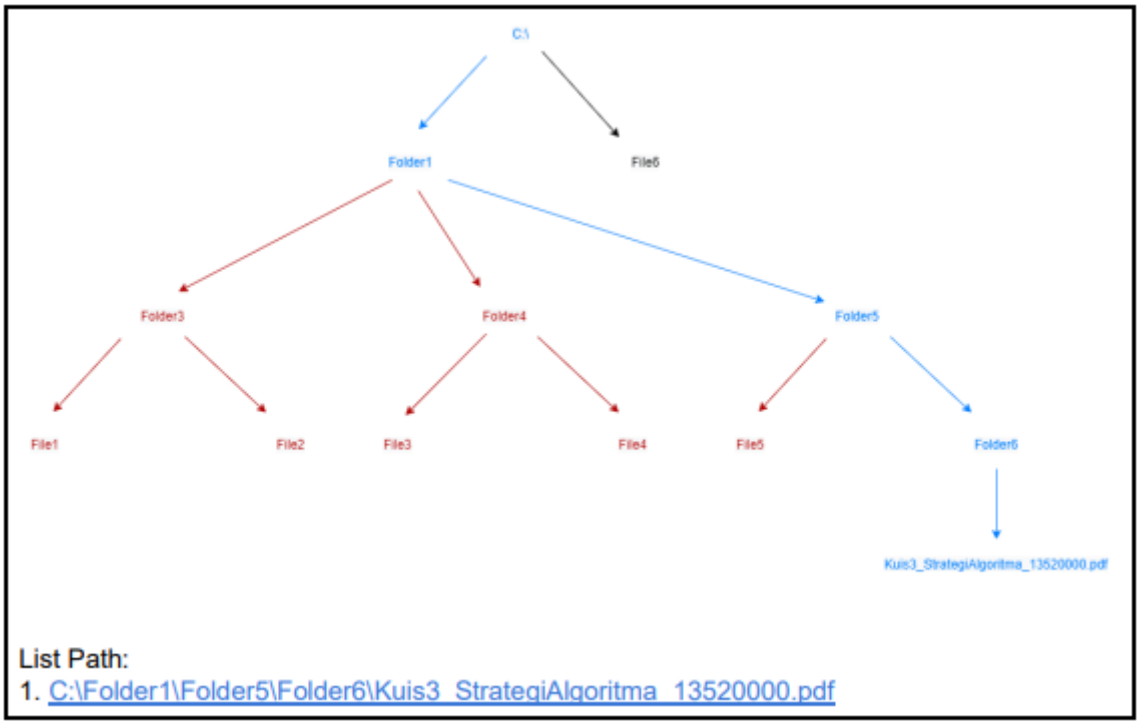
1.1. Deskripsi Tugas

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.

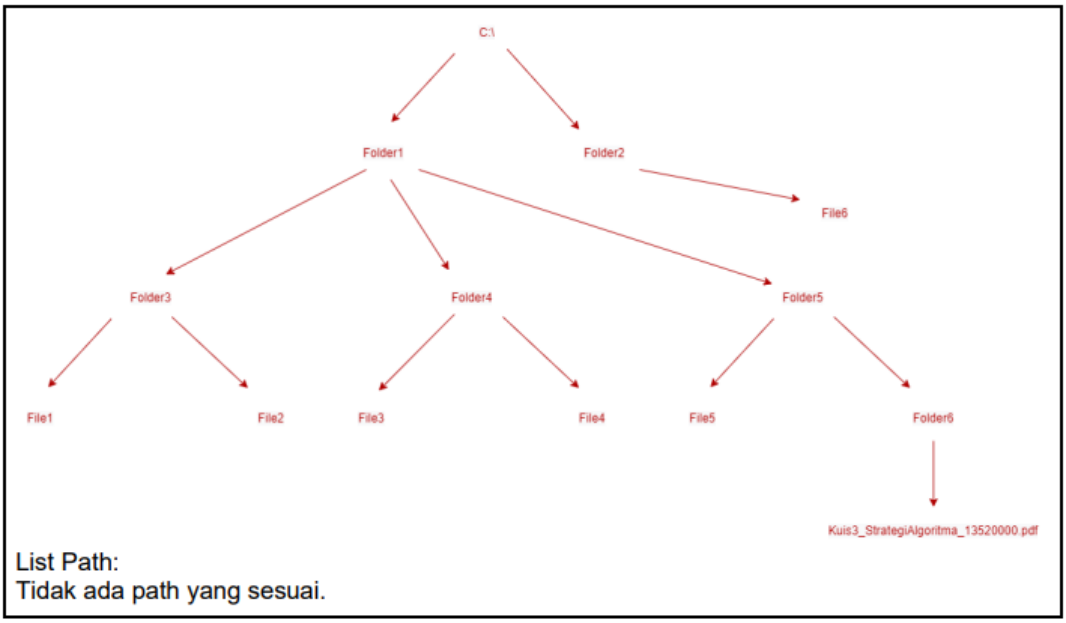


Gambar 1. Contoh input program



Gambar 2. 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.

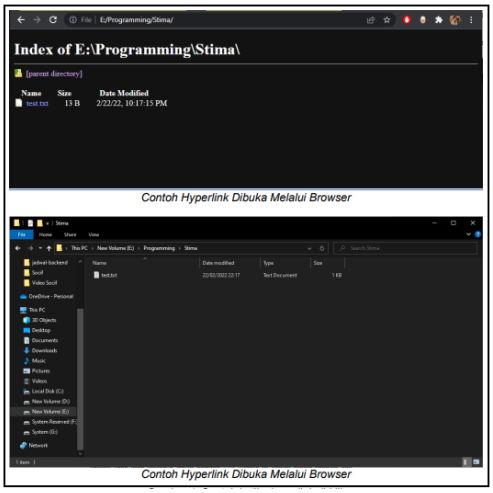


Gambar 3. Contoh output program jika file tidak ditemukan

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 3. Contoh Ketika hyperlink di-klik

BAB 2: Landasan Teori

2.1. Algoritma Traversal Graf

Traversal graf, atau eksplorasi graf tanpa penggantian, merupakan proses algoritma graf dalam mengunjungi tiap simpul dengan cara sistematik. Dalam program kami akan dipakai dua tipe pencarian traversal graf yaitu pencarian melebar (*breadth first search*/BFS) dan pencarian mendalam (*depth first search*/DFS). Pencarian dapat dilakukan jika tiap simpul dalam graf terhubung. Traversal graf tidak akan pernah mengunjungi simpul yang sama lebih dari sekali dan, dengan asumsi semua simpul semuanya terhubungi, akan mengunjungi semua simpul yang ada.

2.2. *Depth First Search*

Depth-first search adalah algoritma traversal graf yang dimulai dari simpul akar dan secara progresif menjelajahi semua tetangga simpul tersebut. Pada setiap iterasi baru, akan dipilih simpul terbaru yang bertetangga dengan yang dieksplorasi tetapi belum dikunjungi, sehingga terlebih dahulu mengeksplorasi simpul dahulu simpul yang terjauh dari simpul akar.

1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v
3. Ulangi DFS mulai dari simpul w
4. Ketima mencapai simpul u sedemikian sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

2.3. *Breadth First Search*

Breadth-first search (BFS) adalah algoritma traversal graf yang dimulai dari simpul akar dan secara progresif menjelajahi semua tetangga simpul tersebut, sama seperti DFS. Pada setiap iterasi baru, simpul yang paling awal ditemukan tetapi belum dikunjungi dipilih berikutnya utuk dikunjungi. Dengan demikian, BFS menemukan semua simpul dalam jarak tertentu dari simpul akar. Algoritma ini bekerja sebagai berikut (anggap simpul akar merupakan simpul *v*):

1. Kunjungi simpul v
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu
3. Kunjungi simpul yang belum dikunjugi dan bertetangga dengan simpul-simpul yang tadi dikunjungi
4. Ulangi tahap ke-3 sampai suatu kondisi terpenuhi.

2.4. C# *Desktop Application Development*

Dalam tugas ini, kami menggunakan aplikasi *desktop* Visual Studio untuk membangun aplikasi dengan bahasa C# kami. Pertama-tama, kita perlu mengunduh dan men-*install* aplikasi tersebut. Kemudian, akan dibuat sebuah C# project ber-*template* Windows Forms App untuk memulai membangun aplikasi. Setelah masuk ke dalam project, kita bisa memulai membangun aplikasi. Dalam *tab* Design, kita bisa mengubah desain form kita, untuk menambahkan suatu elemen, terdapat *tab* Toolbox yang memiliki daftar elemen-elemen yang bisa ditambahkan dalam desain kita. Kita juga bisa mengubah properti-properti elemen yang kita tambahkan di property viwer di kanan bawah. Apabila ingin menambahkan kode kepada projek kita, kita bisa membuat file C# baru di opsi file viewer di kanan atas, atau menambahkan kode di file .cs aplikasi. Apabila kita telah selesai membuat aplikasi dan ingin mengecek jalannya aplikasi, kita bisa memencet tombol *Run* yang ada di bagian atas, dan apabila sudah puas kita bisa mem-*build* aplikasi dengan mengeklik kanan nama projek kita di file viewer, dan mengeklik tombol *build.*

BAB 3: Analisis Pemecahan Masalah

3.1. Langkah-langkah pemecahan masalah

1. Membuat algoritma DFS dan BFS untuk mencari nama file dalam direktori folder
    - Algoritma bekerja dengan cara mengecek semua file/folder dalam direktori folder yang diberikan yang memiliki nama file yang sama dengan input query pengguna
2. Membuat implementasi pembuatan graf dari algoritma DFS dan BFS dengan library MSAGL
3. Implementasi pilihan pencarian terhadap DFS dan BFS
    - Membuat implementasi pilihan pencarian dalam DFS dan BFS yaitu mencari 1 file saja serta mencari semua kemunculan file pada folder root.
4. Membuat GUI dalam C# Desktop Application Development
    - Membuat implementasi input query nama file/folder, input direktori awal folder, input pemilihan algoritma pencarian (BFS/DFS), serta input pemilihan pencarian (1 file/semua kemunculan file)
    - Penggunaan implementasi algoritma pembuatan graf dari algoritma BFS/DFS berdasarkan query yang dimasukkan dalam GUI
    - Menghitung waktu yang dibutuhkan untuk algoritma pencarian BFS/DFS selesai.
    - Menampilkan waktu jalan program, hasil graf, dan hasil pencarian menggunakan algoritma BFS/DFS berupa list hyperlink *directory* folder
    - Hyperlink *directory* folder dibuka menggunakan file browser

3.2. Mapping Elemen Algoritma

- Simpul (*Problem State*): Semua folder/file dalam direktori folder yang dipilih
- Simpul Awal (*Initial State*): Direktori folder yang dipilih
- Daun (*Solution State*): Semua file dalam path folder awal yang bersesuaian dengan nama input query file
- Ruang Solusi: Himpunan semua path ditemukannya nama file yang bersesuaian

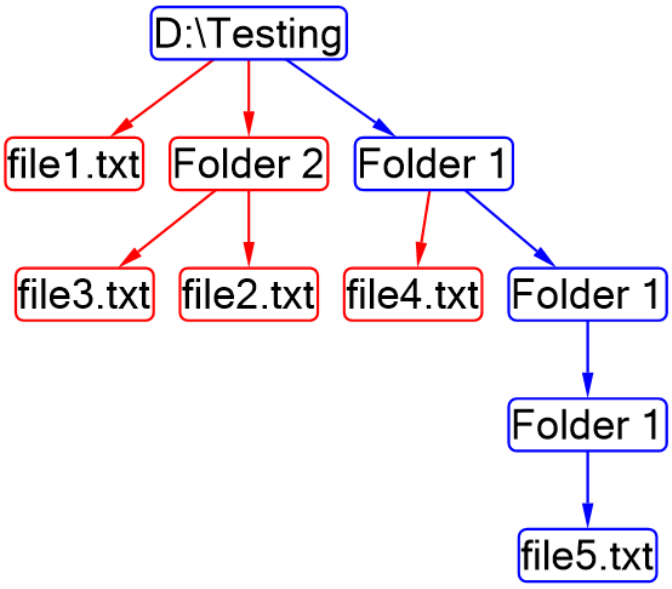
3.3. Ilustrasi Kasus Lain

3.3.1. Kasus lain 1: Nama Folder/File yang Sama

Jika terdapat beberapa nama folder/file yang sama tetapi berada pada direktori yang berbeda, maka nama folder/file tersebut akan dibuat simpul baru pada visualisasi graf. Contoh visualisasi kasus:

- Input query: file1.txt
- Direktori folder: D:\Testing
- Pilihan pencarian: BFS, mencari semua kemunculan file

- Output graf:



Gambar 4. Contoh Visualisasi Kasus Lain 1

BAB 4: Implementasi Dan Pengujian

4.1. Implementasi Program

Berikut merupakan implementasi fungsi algoritma BFS dan DFS terhadap pencarian direktori file serta implementasi pembuatan graf:

4.1.1. Algoritma BFS dalam `Program.cs`

```
function BFS(find_all_occurence: boolean)

KAMUS CLASS (class Program)
find_file: string
root_path: string
path_list: List<string>
file_count: Dictionary<string, int>
list_graph: List<(string,string)>
programRunTime: Stopwatch

KAMUS LOKAL
found: boolean
queue_BFS: Queue<(string, string)>
directories: Array of string
files: Array of string
current_queue: (string, string)
parent_path: string
temp_name: string

ALGORITMA
{ Belum ketemu }
found ← false
{ Membersihkan graf dan list direktori hasil pencarian sebelumnya }
path_list.Clear()
list_graph.Clear()
file_count.Clear()
{ Mencari semua file dan folder dalam direktori yang dipilih lalu
menambahkannya ke dalam queue BFS dalam bentuk tuple }
directories ← Directory.GetDirectories(root_path)
files ← Directory.GetFiles(root_path)
i traversal [0..directories.Neff]
     queue_BFS.Enqueue((root_path, directories[i]))
i traversal [0..files.Neff]
     queue_BFS.Enqueue((root_path, files[i]))
```

```
{ Melakukan iterasi queue BFS sampai habis untuk skema BFS }
while (queue_BFS.Count > 0) do
      current_queue ← queue_BFS.Dequeue()
      parent_path ← current_queue.Item1
      current_path ← current_queue.Item2
      file_name ← Path.GetFileName(current_path)
      temp_name ← file_name { Menyimpan sementara nama file yang akan
      dicek }
      { Jika terdapat nama file/folder yang sudah ada sebelumnya, akan
      dibuat nama baru }
      if not(file_count.ContainsKey(file_name)) then
            file_count[file_name] = 0
            file_name
      else
            file_count[file_name]++
            file_name += "->" + file_count[file_name]

      if (temp_name ≠ find_file) then
            { Jika current_path merupakan folder, tambahkan semua
            file/folder (tetangga) dalam folder tersebut ke dalam
            queue BFS }
            if not found then
                  if (Directory.Exists(current_path)) then
                        directories ←
                              Directory.GetDirectories(current_path)
                        files ← Directory.GetFiles(current_path)
                        i traversal [0..directories.Neff]
                              queue_BFS.Enqueue((file_name, directory))
                        i traversal [0..files.Neff]
                              queue_BFS.Enqueue((file_name, file))
      else
            { Jika nama file sesuai dengan input query, direktori file
            tersebut dimasukkan ke dalam path_list }
            if not(found) then
                  this.path_list.Add(Directory.
                        GetParent(current_path).FullName)
            if not(find_all_occurence) and not(found) then
            { Jika pemilihan pencarian adalah 1 file saja, runtime
            dihentikan }
                  found ← true
                  this.programRunTime.Stop()
```

```
        { Menambahkan file/folder dan direktorinya ke dalam list_graph
        untuk penggambaran graf }
        list_graph.Add((parent_path, file_name))
{ Jika pemilihan pencarian adalah all find occurance runtime dihentikan
ketika queue BFS sudah habis seluruhnya }
if (this.programRunTime.IsRunning) then
        this.programRunTime.Stop()
```

4.1.2. Algoritma DFS dalam `Program.cs`

```
function DFS(find_all_occurence: boolean)

KAMUS CLASS (class Program)
find_file: string
root_path: string
path_list: List<string>
file_count: Dictionary<string, int>
list_graph: List<(string,string)>
programRunTime: Stopwatch

KAMUS LOKAL
found: boolean
stack_DFS: Queue<(string, string)>
directories: Array of string
files: Array of string
current_queue: (string, string)
parent_path: string
temp_name: string

ALGORITMA
{ Belum ketemu }
found ← false
{ Membersihkan graf dan list direktori hasil pencarian sebelumnya }
path_list.Clear()
list_graph.Clear()
file_count.Clear()
{ Mencari semua file dan folder dalam direktori yang dipilih }
directories ← Directory.GetDirectories(root_path)
files ← Directory.GetFiles(root_path)
{ memasukkan file dan folder hasil pencarian ke dalam stack
DFSmenambahkannya ke dalam stack DFS dalam bentuk tuple. Untuk
pengambilannya dimulai dari belakang (z-a) tujuannya adalah agar urutan
pengambilan stack DFS tetap a-z }
i traversal [files.Neff..0]
      stack_DFS.Push((root_path, files[i]))
i traversal [directories.Neff..0]
      stack_DFS.Push((root_path, directories[i]))
{ Melakukan iterasi stack DFS sampai habis untuk skema DFS }
while (stack_DFS.Count > 0) do
      current_stack ← stack_DFS.Dequeue()
      parent_path ← current_stack.Item1
```

```
      current_path ← current_stack.Item2
      file_name ← Path.GetFileName(current_path)
      temp_name ← file_name { Menyimpan sementara nama file yang akan
      dicek }
      { Jika terdapat nama file/folder yang sudah ada sebelumnya, akan
      dibuat nama baru }
      if not(file_count.ContainsKey(file_name)) then
            file_count[file_name] = 0
            file_name
      else
            file_count[file_name]++
            file_name += "->" + file_count[file_name]

      if (file_count[file_name] ≠ 0) then
            file_name += "->" + file_count[file_name]

      if (temp_name = find_file) then
            { Jika nama file sesuai dengan input query, direktori file
            tersebut dimasukkan ke dalam path_list }
            if not(found) then
                  this.path_list.Add(Directory.
                        GetParent(current_path).FullName)
            if not(find_all_occurence) and not(found) then
            { Jika pemilihan pencarian adalah 1 file saja, runtime
            dihentikan }
                  found ← true
                  this.programRunTime.Stop()
      else
            { Jika current_path merupakan folder, tambahkan semua
            file/folder (tetangga) dalam folder tersebut ke dalam
            stack DFS (file/folder yang dimasukkan menggunakan
            aturan yang sama dengan sebelumnya }
            if not found then
                  if (Directory.Exists(current_path)) then
                        directories ←
                              Directory.GetDirectories(current_path)
                        files ← Directory.GetFiles(current_path)
                        i traversal [files.Neff..0]
                              stack_DFS.Push((root_path, files[i]))
                        i traversal [directories.Neff..0]
                              stack_DFS.Push((root_path, directories[i]))
                  { Menambahkan file/folder dan direktorinya ke dalam
```

```
            list_graph untuk penggambaran graf }
            list_graph.Add((parent_path, file_name))
{ Jika pemilihan pencarian adalah all find occurance runtime dihentikan
ketika stack DFS sudah habis seluruhnya}
if (this.programRunTime.IsRunning) then
      this.programRunTime.Stop()
```

4.1.3. Algoritma Gambar Graf

```
function animateGraph(find_all_occurence: boolean)

KAMUS CLASS (class Program)
find_file: string
root_path: string
path_list: List<string>
file_count: Dictionary<string, int>
list_graph: List<(string,string)>

KAMUS CLASS (class GUI)
gViewer1: GViewer
animation_speed: int

KAMUS LOKAL
found: boolean
graph: Graph
node_children: Node
edge: Edge
list_edge: List<Edge>
parent_file_name: string
node_id_parent: string
children_file_name: string
node_id_children: string
str_parent: array of string
str_children: array of string
check_parent: string
check_children: string

ALGORITMA
{ Belum ketemu }
found ← false
{ memberi warna merah pada node root yang menunjukan sudah dilewati }
graph.AddNode(root_path).Attr.Color ←
```

```
                                        Microsoft.Msagl.Drawing.Color.Red

{ iterasi satu per satu setia list_graph yang sudah disimpan dari hasil
pencarian }
i traversal [0..list_graph.Neff]
      parent_file_name ← list_graph[i].Item1
      node_id_parent ← parent_file_name
      children_file_name ← list_graph[i].Item2
      node_id_children ← children_file_name
      str_parent ← parent_file_name.Split("->")
      str_children ← children_file_name.Split("->")
      { apabila array str_parent atai str_children memiliki panjang
      lebih dari dua berarti ada file/ folder dengan nama sama untuk
      node parent atau children, maka parent_file_name/
      children_file_name akan berbeda dengan node_idnya }
      if (str_parent.Length > 1) then
            parent_file_name ← str_parent[0]
            node_id_parent ← str_parent[1]
      if (str_children.Length > 1) then
            children_file_name ← str_children[0]
            node_id_children ← str_children[1]
      { membuat node_children }
      node_children ← Node(children_file_name)
      node_children.Id ← node_id_children
      { kasus pewarnaan node_children biru = ketemu, merah = sudah
      Dilewati, hitam = ada diantrian namun belum dilewati }
      if not found then
            { visualisasi graph yang sudah terbentuk untuk iterasi saat
            itu }
            gViewer.Graph = graph
            await Task.Delay(animation_speed)
      if (children_file_name == find_file)
            node_children.Attr.Color ←
                              Microsoft.Msagl.Drawing.Color.Blue
      else
            if not found then
                  node_children.Attr.Color ←
                                    Microsoft.Msagl.Drawing.Color.Red
            else
                  node_children.Attr.Color ←
                                    Microsoft.Msagl.Drawing.Color.Black
      { menambah node_children ke graph }
```

```
graph.AddNode(node_children)
{ kasus pewarnaan edge biru = ketemu, merah = sudah
Dilewati, hitam = ada diantrian namun belum dilewati }
{ visualisasi graph ketika root_path sudah terbentuk }
gViewer.Graph = graph
await Task.Delay(animation_speed)
edge ← Edge(graph.FindNode(node_id_parent), node_children,
                                    ConnectionToGraph.Connected
if (children_file_name == find_file)
     edge.Attr.Color ← Microsoft.Msagl.Drawing.Color.Blue
else
     if not found then
          edge.Attr.Color ← Microsoft.Msagl.Drawing.Color.Red
     else
          edge.Attr.Color ← Microsoft.Msagl.Drawing.Color.Black
{ menambah edge ke list_edge }
list_edge.Add(edge)
{ jika file_name_children dari list graph sama dengan file yang
dicari }
if (children_file_name == find_file) then
     { melakukan penesuluran di list_edge yang sudah terbentuk
     Untuk mengubah warna edge dan node yang tadinya berwarna
     Merah (sudah dilewati) menjadi biru (path ini menuju file
     yang dicari) }
     if not found then
          check_parent ← node_id_children
          repeat
               i traversal [0..list_edge.Neff]
                    if list_edge[i].TargetNode.Id =
                       check_parent then
                          check_children ← check_parent
                          check_parent ←
                                    list_edge[i].SourceNode.Id
                          list_edge[i].Attr.Color ←
                          Microsoft.Msagl.Drawing.Color.Blue
                          list_edge[i].SourceNode.Attr.Color ←
                          Microsoft.Msagl.Drawing.Color.Blue
               { visualisasi graph terhadap perubahan edge dan
               node yang tadinya berwarna merah menjadi berwarna
               biru }
               gViewer.Graph = graph
               await Task.Delay(animation_speed)
```

```
            until check_parent = root_path
        { Jika pemilihan pencarian adalah 1 file saja, node dan edge
        yang terbentuk setelah node ini berwarna hitam }
        if not found and not find_all_occurance then
            found ← true
    { visualisasi graph yang sudah lengkap terbentuk }
    gViewer.graph ← graph
```

4.2. Struktur Data Program

Struktur data yang digunakan untuk menyelesaikan program *folder crawling* yaitu *queue*, *stack*, *dictionary*, *list*, *tuple*, dan *class*. *Queue* digunakan untuk menyimpan simpul hidup serta simpul ekspan yang dibangkitkan untuk algoritma BFS. *Stack* digunakan untuk menyimpan simpul hidup serta simpul ekspan yang dibangkitkan untuk algoritma DFS. Penggunaan *queue* juga memudahkan untuk melakukan *backtrack*. Kemudian *dictionary* digunakan untuk mengatasi penggunaan nama file/folder yang sama untuk mempermudah pembuatan graf mengguanakan library MSAGL. *List* digunakan untuk menyimpan dua simple dalam *tuple* untuk representasi senarai sisi graf. *Tuple* digunakan untuk menggambarkan suatu sisi dari dua simpul yang berhubungan. Penggunaan *class* akan digambarkan sebagai diagram kelas untuk berikut:

Gui
- dirChosen: string - hasDir: boolean - hasFilename: boolean - findAllOccurence: boolean - p: Process
- Main() + Gui() - dirButton_Click(object sender, EventArgs e) - textBox1_TextChanged(object sender, EventArgs e) - checkBox1_CheckedChanged(object sender, EventArgs e) - button1_Click(object sender, EventArgs e) - animateGraph(Boolean find_all_occurence) - showResultPath(List<string> resultPath) - richTextBox1_LinkClicked(object sender, System.Windows.Forms.LinkClickedEventArgs e)

Program
- find_file: string - root_path: string - path_list: List<string> - file_count: Dictionary - list_graph: List - programRunTime: Stopwatch
+ Program(string find_file, string rooth_path) + BFS(bool find_all_occurence) + DFS(bool find_all_occurence) + elapsedTime(): string

4.3. Tata Cara Penggunaan Program

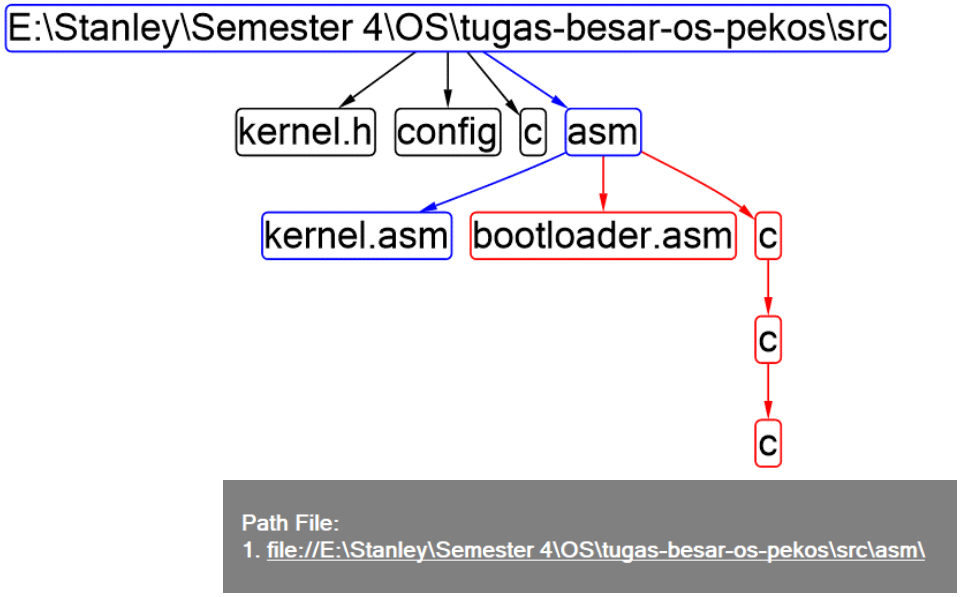
1. Jalankan aplikasi FolderCrawling.exe dalam folder bin. Tampilan menu utama aplikasi terlihat sebagai berikut:
2. Tekan tombol Choose Directory untuk memilih direktori awal pencarian
3. Masukkan nama file pada TextBox “Input Filename” untuk input query nama file yang ingin dicari
4. Centang tombol checkbox “Find All Occurrence” jika ingin mencari semua direktori nama file yang sama dengan input query. Jika hanya ingin mendapatkan direktori 1 nama file saja, checkbox tidak perlu dicentang.
5. Pilih algoritma yang ingin digunakan pada RadioButton (BFS dan DFS).
6. Tekan tombol “START” untuk memulai pencarian nama file.

4.4. Hasil Pengujian

4.4.1. Pengujian 1 : File yang Di Cari Ditemukan

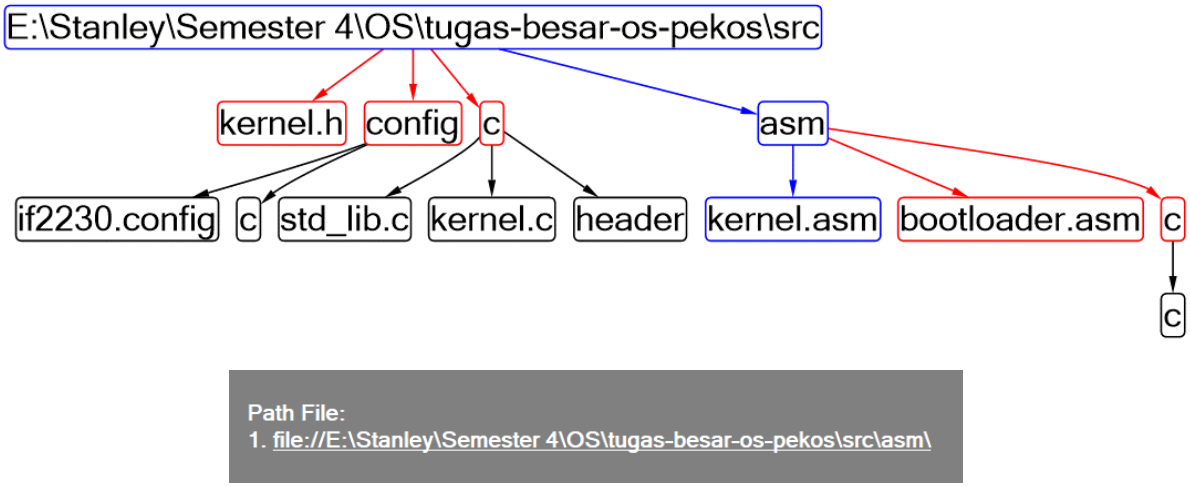
Jika file yang di cari berhasil temukan. Path menuju file tersebut akan diberi warna biru. Path-path lain yang sudah dilewati dalam pencarian file akan diberi warna merah, sedangkan untuk path-path lain yang belum dilewati diberi warna hitam. Selain itu, path yang ditemukan juga akan ditampilkan dalam bentuk *hyperlink* Contoh visualisasi kasus:

- Input query: kernel.asm
- Direktori folder: E:\Stanley\Semester 4\OS\tugas-besar-os-pekos\src
- Pilihan pencarian: DFS, mencari kemunculan file pertama



Gambar 5. Contoh Pengujian 1 (1)

- Input query: kernel.asm
- Direktori folder: E:\Stanley\Semester 4\OS\tugas-besar-os-pekos\src
- Pilihan pencarian: BFS, mencari kemunculan file pertama

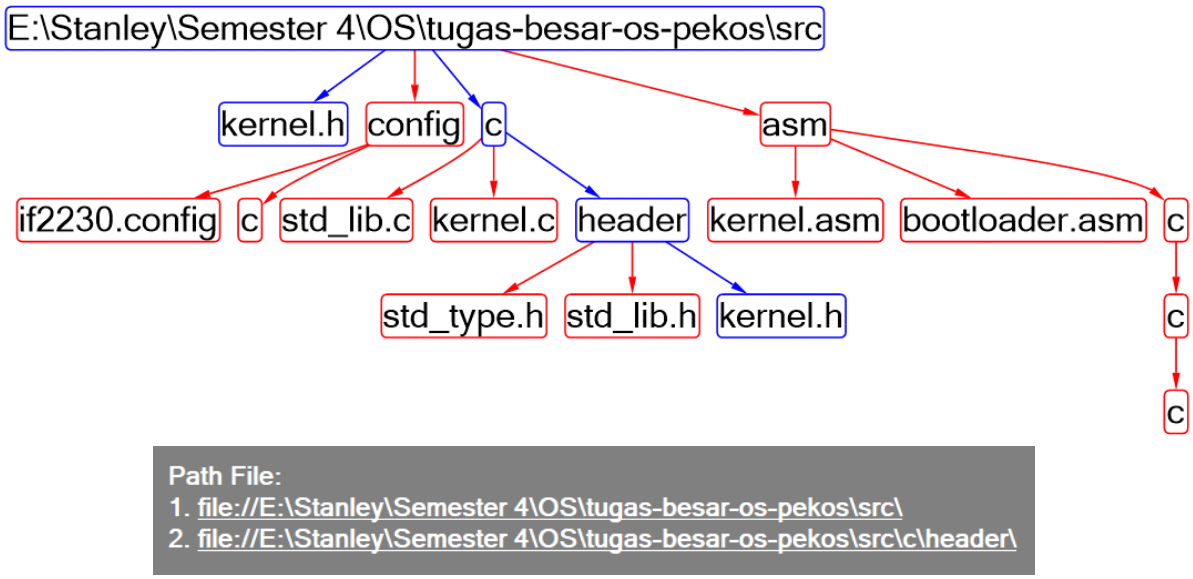


Gambar 6. Contoh Pengujian 1 (2)

4.4.2. Pengujian 2: File yang Di Cari Memiliki Lebih dari Satu Nama yang Sama

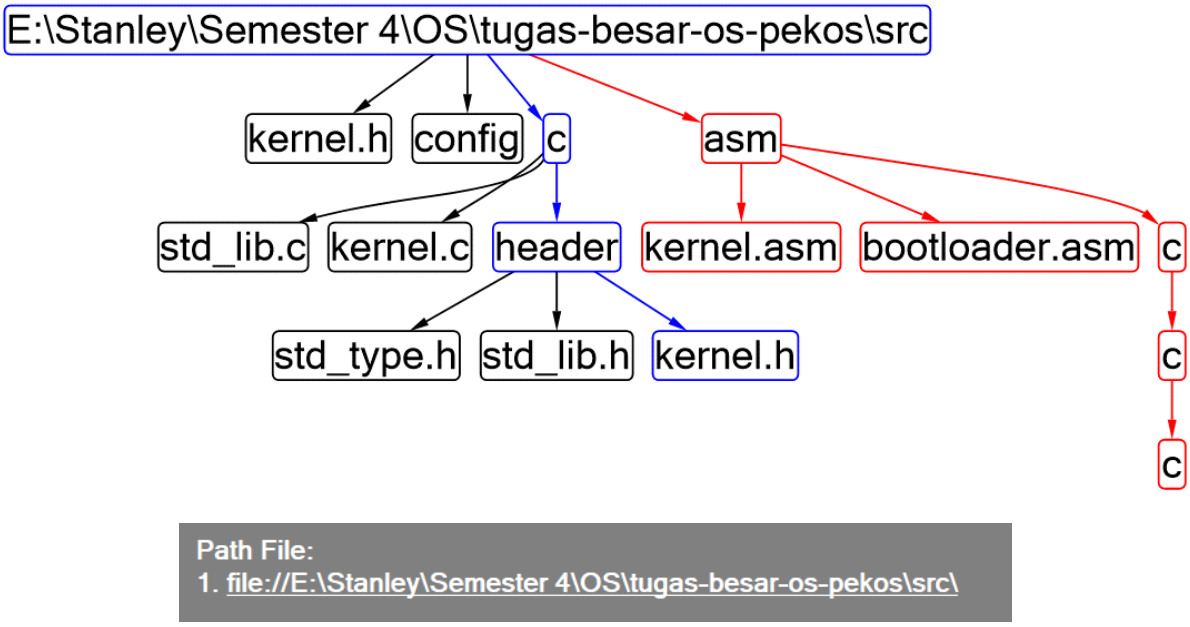
Jika file yang dicari memiliki nama file yang sama tetapi berada pada direktori yang berbeda, maka jika user memiliki find_all_occurance maka path-path yang menuju file-file tersebut akan diwarnai biru, jika tidak hanya path yang menuju file pertama hasil pencarian yang akan diwarnai biru. Selain itu, path-path yang ditemukan juga akan ditampilkan dalam bentuk *hyperlink*, path-path yang ditampilkan juga mengikuti aturan find_all_occurance atau tidak. Contoh visualisasi kasus:

- Input query: kernel.h
- Direktori folder: E:\Stanley\Semester 4\OS\tugas-besar-os-pekos\src
- Pilihan pencarian: DFS /BFS, mencari semua kemunculan file



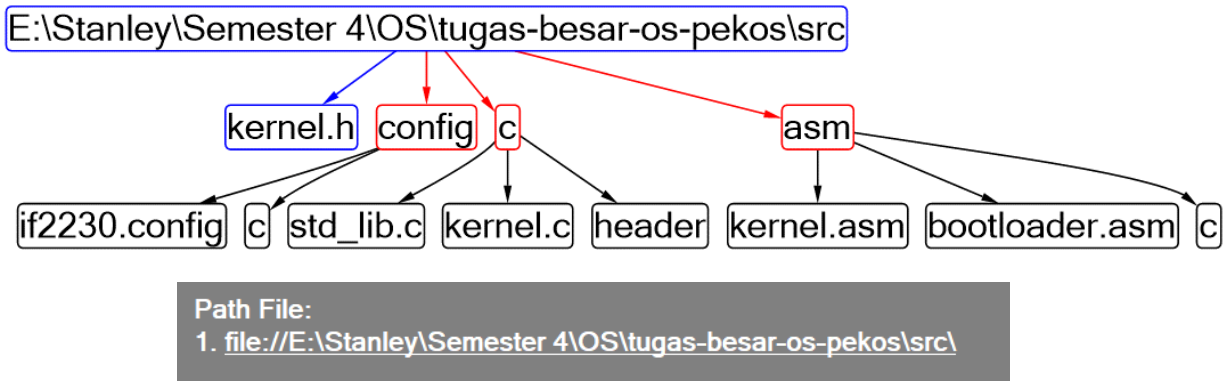
Gambar 7. Contoh Pengujian 2 (1)

- Input query: kernel.h
- Direktori folder: E:\Stanley\Semester 4\OS\tugas-besar-os-pekos\src
- Pilihan pencarian: DFS, mencari kemunculan file pertama



Gambar 8 Contoh Pengujian 2 (2)

- Input query: kernel.h
- Direktori folder: E:\Stanley\Semester 4\OS\tugas-besar-os-pekos\src
- Pilihan pencarian: BFS, mencari kemunculan file pertama

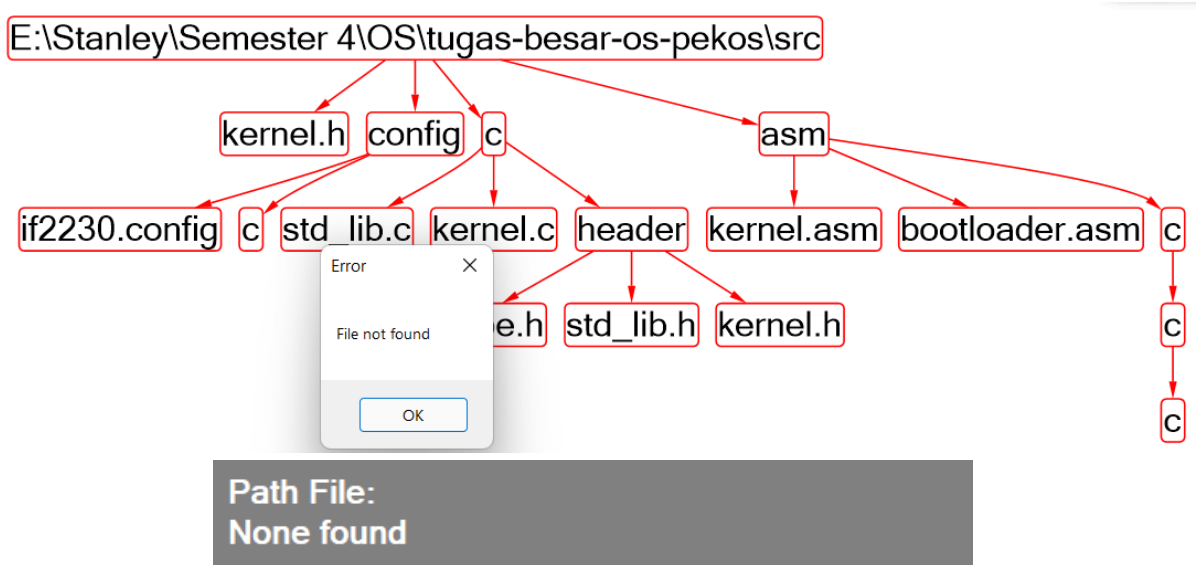


Gambar 9. Contoh Pengujian 2 (3)

4.4.3. Pengujian 3: File yang Di Cari Tidak Ditemukan

Jika file yang dicari tidak berhasil ditemukan, setelah graf terbentuk dengan sempurna, akan muncul pesan error yang mengatakan bahwa file yang dicari tidak ditemukan. Contoh visualisas kasus :

- Input query: kernel.ht
- Direktori folder: E:\Stanley\Semester 4\OS\tugas-besar-os-pekos\src
- Pilihan pencarian: DFS /BFS, mencari semua kemunculan file

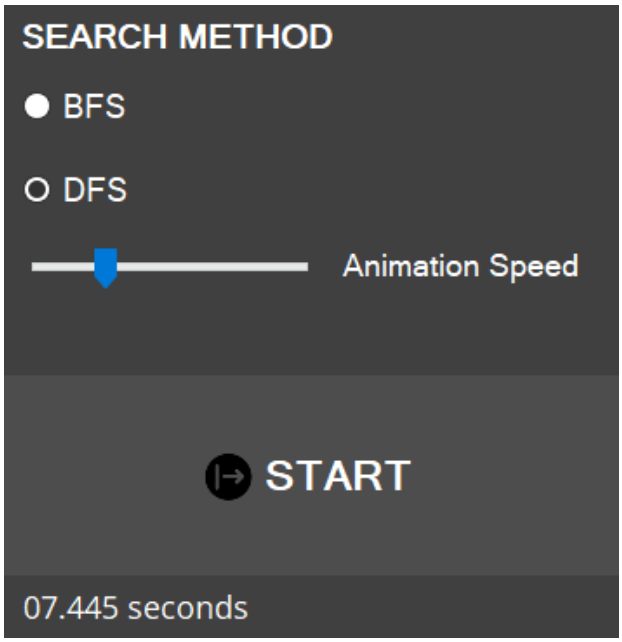


Gambar 10. Contoh Pengujian 3

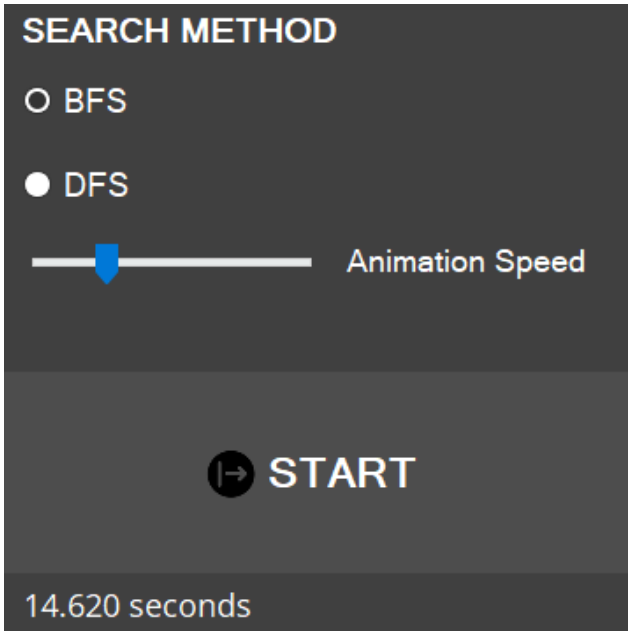
4.4.4. Pengujian 4: Average Case dalam folder akar yang mempunyai kedalaman besar dan folder anak sedikit

Untuk merepresentasikan *average case* dalam kasus yang mana suatu folder akar memiliki kedalaman besar dan folder anak sedikit, dibuat suatu file di dalam folder dummy berisi 3 anak, di dalam suatu folder dengan 2 folder yang lain yang memiliki hal yang sama tanpa file tersebut, di dalam suatu folder dengan 2 folder lain yang memiliki hal yang sama, terus menerus sedemikian rupa sehingga terdapat 3^9 folder dan 1 file. Karena jumlah folder yang tinggi, akan diambil waktu eksekusi algoritma pencarian saja sebagai perbandingan kecepatan karena graf yang dibentuk tidak dapat dibaca.

- Input query: findme
- Direktori folder: C:\Lenovo%20Temp\IF%20Files\Sem4\Stima\Testing\KedalamanBesarAnakDikit
- Pilihan pencarian: BFS & DFS, mencari kemunculan file pertama



Gambar 11. Contoh Pengujian 4 (1)

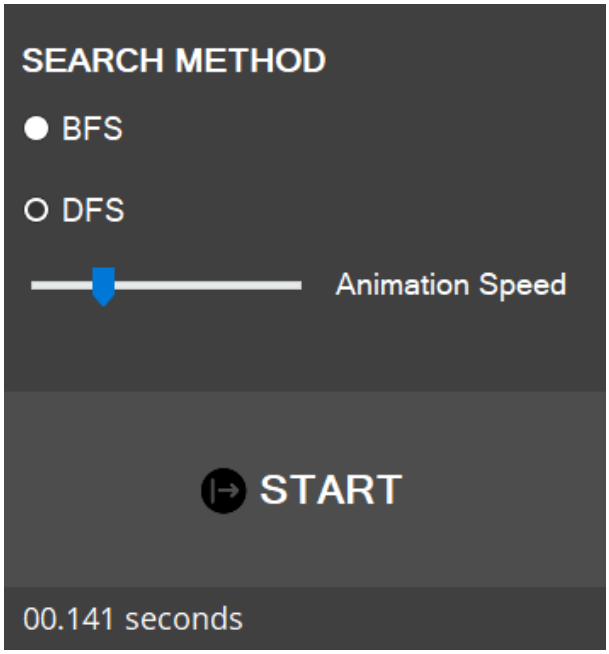


Gambar 12. Contoh Pengujian 4 (2)

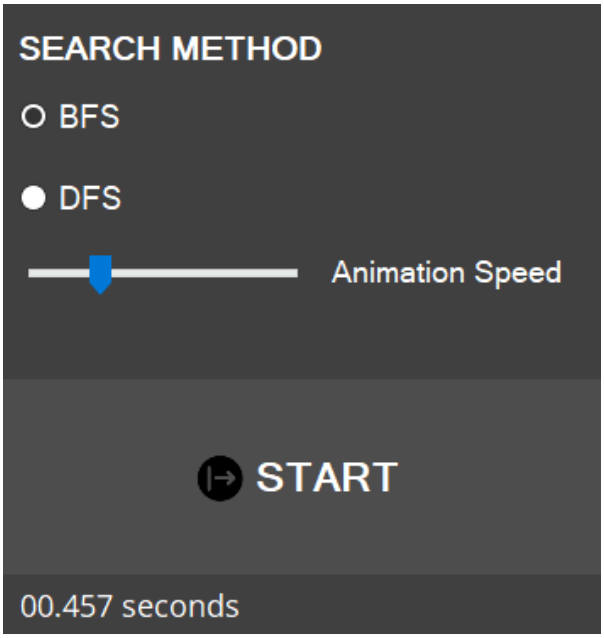
4.4.5. Pengujian 5: Average Case dalam folder akar yang mempunyai kedalaman kecil dan folder anak banyak

Untuk merepresentasikan *average case* dalam kasus yang mana suatu folder akar memiliki kedalaman rendah dan folder anak banyak, dibuat suatu file di dalam folder dummy berisi 9 anak, di dalam suatu folder dengan 8 folder yang lain yang memiliki hal yang sama tanpa file tersebut, di dalam suatu folder dengan 8 folder lain yang memiliki hal yang sama sehingga terdapat 9^3 folder dan 1 file. Karena jumlah folder yang tinggi, akan diambil waktu eksekusi algoritma pencarian saja sebagai perbandingan kecepatan.

- Input query: findme
- Direktori folder: C:\Lenovo%20Temp\IF%20Files\Sem4\Stima\Testing\KedalamanPendekAnakBanyak
- Pilihan pencarian: BFS & DFS, mencari kemunculan file pertama



Gambar 13. Contoh Pengujian 5 (1)



Gambar 14. Contoh Pengujian 5 (2)

4.5. Analisis Desain Solusi Algoritma BFS dan DFS

Apabila kita melihat cara kerja BFS dan DFS, tidak ada di antara keduanya yang lebih unggul dibandingkan lawannya. BFS dapat unggul dalam pencarian yang mana suatu folder akar ataupun folder yang sedang dicek algoritma memiliki folder anak yang memiliki kedalaman tinggi. Sedangkan dengan DFS dapat unggul dalam pencarian yang mana folder akar ataupun folder yang sedang dicek algoritma memiliki banyak folder anak, tetapi memiliki kedalaman yang rendah.

Dari hasil pengujian-pengujian yang kami lakukan, kami dapatkan bahwa antara algoritma BFS dan DFS, DFS unggul dalam kasus folder akar ataupun folder yang sedang dicek algoritma memiliki folder anak yang memiliki kedalaman tinggi dan kasus folder akar ataupun folder yang sedang dicek algoritma memiliki banyak folder anak, tetapi memiliki kedalaman yang rendah. Kedua algoritma mampu menghasilkan path ke file yang ditentukan.

BAB 5: Kesimpulan dan Saran

5.1. Kesimpulan

Kami berhasil untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari file explorer pada sistem operasi yaitu pencarian file yang dalam program disebut sebagai *Folder Crawling*. Fitur menggunakan implementasi algoritma BFS atau DFS untuk mencari file atau folder yang ditetapkan. Diberikan opsi untuk mencari satu saja kemunculan file/folder atau semua kemunculan. Program telah dapat memvisualisasikan hasil dari pencarian dalam bentuk pohon serta menampilkan hasil *path* ke folder yang memiliki file/folder yang ditetapkan dalam bentuk list berisi *hyperlink* sehingga pengguna bisa menekan *hyperlink* tersebut untuk membuka *file explorer* ke *path* tersebut.

5.2. Saran

Apabila waktu memungkinkan, dapat diperbagus sistem *hyperlink* karena program kami masih menggunakan imbuhan `file://` sebelum *path* file/folder agar dapat dikenal sebagai suatu *link* dari sistem.

Daftar Pustaka

Microsoft. *Create a Windows Forms app in Visual Studio with C#*. 2022, https://docs.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022awdsaas

Kurant, Maciej et al. *On The Bias Of BFS*. 2022, https://www.researchgate.net/profile/Maciej-Kurant/publication/45911249_On_the_bias_of_BFS/links/00b4952b8d4773b4e1000000/On-the-bias-of-BFS.pdf

Munir, Rinaldi. Slide Pengajaran IF221 Strategi Algoritma. 2022, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/stima21-22.htm

Lampiran Link

Link *Source Code* (Github):

https://github.com/Stanley77-web/TubesStima2

Link Video Program (Youtube):

https://youtu.be/P_DjN9N9rew