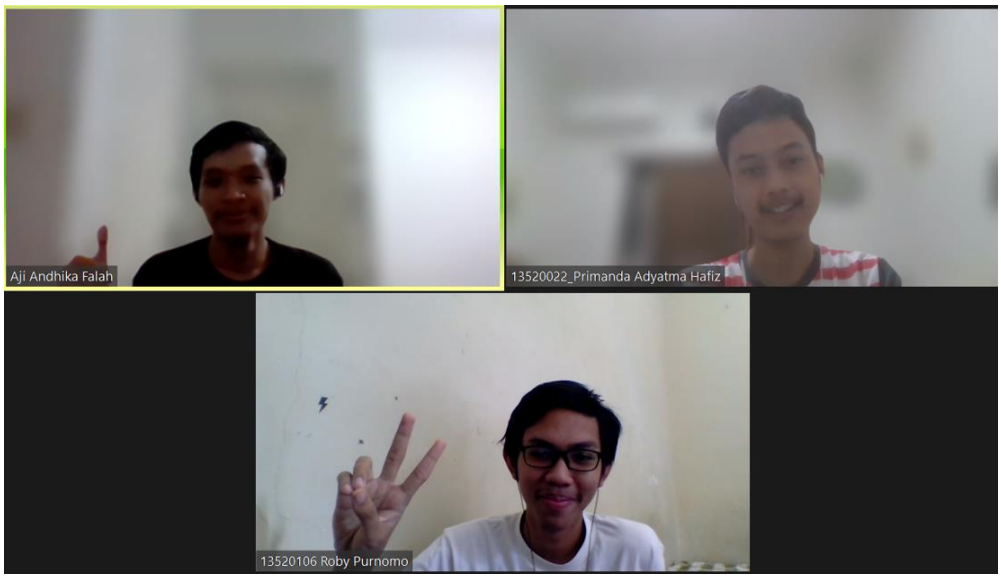
# Laporan Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder* Crawling

TUGAS BESAR STRATEGI ALGORITMA

Oleh:

**Aji Andika Falah - 13520012**
**Primanda Adyatma Hafiz - 13520022**
**Roby Purnomo – 13520106**




**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2022**

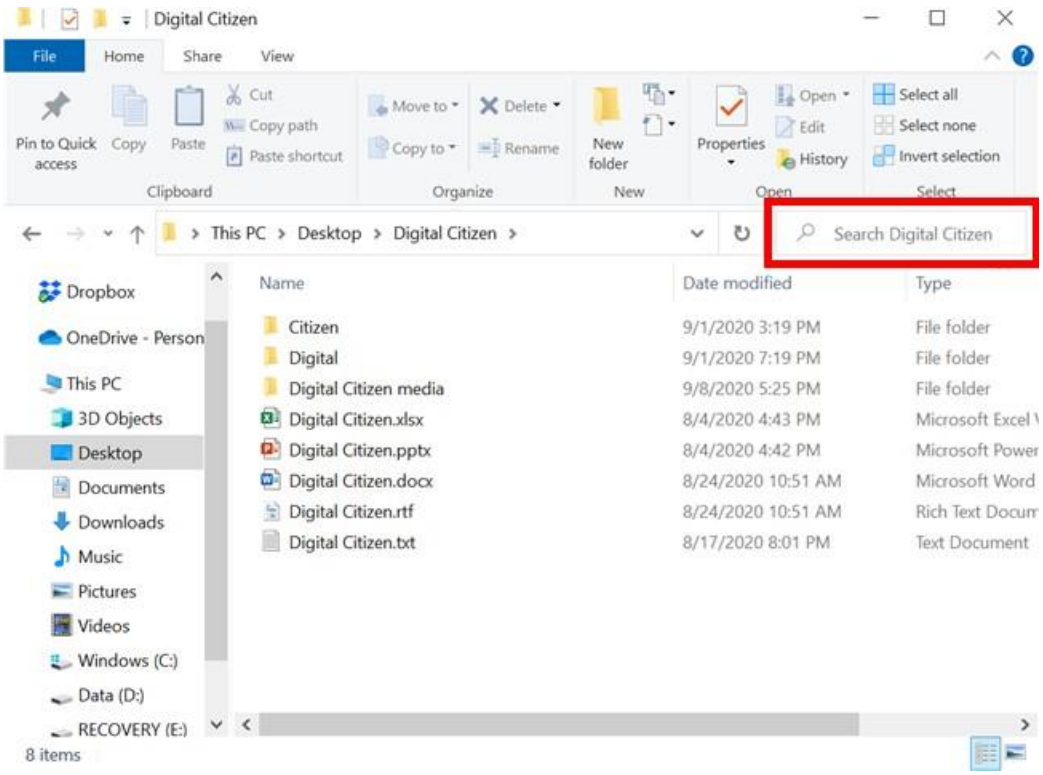
# Daftar Isi

Daftar Isi ..... 1
BAB 1 ..... 2
BAB 2 ..... 4
2.1 Dasar Teori ..... 4
2.2 C# Desktop Development ..... 4
BAB 3 ..... 6
3.1 Langkah-langkah Pemecahan Masalah ..... 6
3.2 *Mapping* Persoalan ..... 6
3.2.1 Algoritma *Breadth First Search* (BFS) ..... 6
3.2.2 Algoritma *Depth First Search* (DFS) ..... 7
3.3 Ilustrasi Kasus ..... 7
3.2.1 Algoritma *Breadth First Search* (BFS) ..... 8
3.2.2 Algoritma *Depth First Search* (DFS) ..... 8
BAB 4 ..... 9
4.1 Implementasi Program ..... 9
4.2 Struktur Data yang Digunakan ..... 12
4.3 Tata Cara Penggunaan Program ..... 13
4.4 Hasil Pengujian dan Analisisnya ..... 14
BAB 5 ..... 18
5.1 Kesimpulan dan Saran ..... 18
DAFTAR PUSTAKA ..... 19
LAMPIRAN ..... 20

# 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 1.1 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 *child*ren-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 *child*ren 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:**

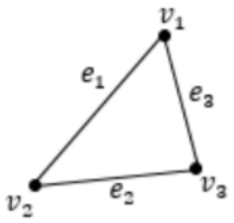
Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari *file* explorer pada sistem operasi, yang pada tugas ini disebut dengan *Folder* Crawling. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Anda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

# BAB 2

# LANDASAN TEORI

## 2.1 Dasar Teori

Graf adalah pasangan dua himpunan yaitu V himpunan tak kosong disebut simpul dan E yang anggotanya disebut sebagai sisi. Sebagai contoh, terdapat graf $G = (V, E)$ yang memiliki $V = \{v_1, v_2, v_3\}$ dan $E = \{(v_1, v_2), (v_2, v_3), (v_1, v_3)\}$ dapat direpresentasikan sebagai berikut.




Gambar 2.1 Representasi Graf G

Dalam proses pencarian elemen pada graf, antara lain terdapat 2 metode yang dapat digunakan yaitu BFS dan DFS. BFS adalah singkatan dari Bread First Search yang prinsipnya mirip dengan *Queue* dan biasanya digunakan dalam persoalan pencarian jalur. Algoritma dari BFS tergolong sederhana dimana pencarian dimulai dari simpul awal, kemudian dilanjutkan ke seluruh simpul tetangganya. Jika simpul tujuan belum ditemukan, maka perhitungan akan diulang lagi ke masing-masing simpul tetangga dari setiap simpul yang telah diperiksa hingga simpul tujuan ditemukan.

Sedangkan DFS atau Depth First Search adalah metode pencarian pada graf yang menggunakan stack untuk menyimpan simpul yang dikunjungi. DFS adalah metode berbasis tepi dan bekerja secara rekursif dimana simpul dieksplorasi di sepaanjang lintasan. Eksplorasi sebuah simpul akan ditangguhkan segera setelah simpul yang belum dijelajahi cocok dengan simpul yang dicari. DFS sendiri pasti melintasi suatu simpul minimal sekali dan maksimal sekali.

## 2.2 C# Desktop Development

C# adalah bahasa pemrograman sederhana yang digunakan untuk tujuan umum, artinya bahasa pemrograman ini dapat digunakan untuk berbagai kebutuhan antara lain pemrograman server-side pada website, membangun aplikasi desktop ataupun mobil, pemrograman game, dan sebagainya. Selain itu C# juga merupakan bahasa pemrograman yang berorientasi objek, jadi C# juga mendukung fitur class, inheritance, polymorphism, dan encapsulation. Dalam prakteknya C# sangat bergantung dengan framework yang disebut dengan .NET Framework, framework inilah yang nantinya digunakan untuk mengcompile dan menjalankan kode C#.

C# dapat digunakan untuk membuat aplikasi GUI sebagai visual dari program. GUI sendiri adalah aplikasi yang mmiliki tombol, halaman, dan widget lainnya sehigga user dapat berinteraksi dengan aplikasi. Dalam pembuatan GUI pada C#, digunakan fitur Windows Form Designer and Toolbox yang dapat diakses pada saat pembuatan project di IDE Visual Studio. Windows Form adalah kumpulan dari elemen yang reusable dan mengenkapsulasi fungsional dari user interface serta digunakan pada client sided Windows based application.

# BAB 3

# ANALISIS PEMECAHAN MASALAH

## 3.1 Langkah-langkah Pemecahan Masalah

Persoalan yang diberikan pada tubes kali ini cukup jelas yakni melakukan pencarian letak (*path*) terhadap suatu nama *file* atau disebut *folder crawling*. Masalah ini dapat dipecahkan dengan melakukan pengecekan terhadap semua *file* yang terdapat pada *directory* awal yang diberikan. Oleh karena itu untuk menjadikan pengecekannya menjadi lebih terarah, persoalan pengecekan ini menggunakan algoritma *Breadth First Search* (BFS) dan *Depth First Search* (DFS). Algoritma BFS dilakukan dengan membuat sebuah *queue* atau antrean. Pada awal akan di *enqueue directory* awal. Lalu dilakukan *dequeue* jika berupa *folder* maka akan dilakukan *enqueue* terhadap semua *file* dan *folder* di dalamnya, dan jika berupa *file* maka akan dicocokkan dengan nama *file* yang diminta. Sedangkan algoritma DFS dilakukan dengan metode rekursif, sehingga pada tiap *folder* (*rekurens*) akan dipanggil lagi fungsi DFS-nya, dan jika berupa *file* (*basis*) akan dilakukan pencocokan dengan nama *file* yang diminta.

## 3.2 *Mapping* Persoalan

Pada program pencarian *file* (*folder crawling*) ini digunakan 2 algoritma yakni algoritma *Breadth First Search* (BFS) dan *Depth First Search* (DFS) dengan detail algoritmanya sebagai berikut.

### 3.2.1 Algoritma *Breadth First Search* (BFS)

1. Hal yang dilakukan pada pencarian yang pertama kali adalah memasukkan *directory* awal pada *queue* atau antrian untuk pencarian.
2. Lalu melakukan *dequeue*, yaitu mengambil head dari *queue* yang mana pada saat ini berisi *directory* awal dan membuat *directory* awal ini sebagai *root* dari graf dan diwarnai dengan warna hitam karena belum mengalami pengecekan. Lalu program akan masuk ke dalam *loop*.
3. Dalam *loop* tersebut, pertama-tama akan dicek apakah head dari *queue* merupakan *file* atau *directory*."
4. Lalu akan dilakukan *dequeue* untuk headnya, jika head merupakan *file* maka akan dicocokkan dengan nama *file* yang dicari, jika *file* sesuai maka *node* tersebut akan diubah ke warna biru beserta dengan *node parent* hingga *root*, dan jika tidak sesuai maka *node* akan menjadi warna merah. Apabila hanya ingin ditampilkan 1 *file* saja yang sesuai, maka program akan dilakukan break sehingga berhenti. Apabila ingin ditampilkan semua *file* yang sesuai, maka program akan berlanjut.
5. Jika head dari *queue* merupakan *file* maka akan di-*list* untuk semua *file* atau *directory* yang ada pada *directory* awal ini dan dimasukkan kedalam *queue* (*enqueue*). Selain

itu, akan dilakukan juga penambahan *node edge* ke dalam graf dan diwarnai dengan warna hitam karena belum mengalami pengecekan.

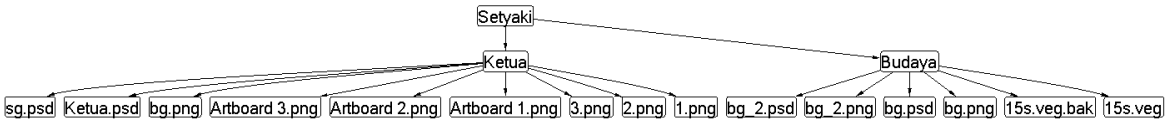
6. *Loop* akan dilakukan hingga *queue* kosong.

### 3.2.2 Algoritma *Depth First Search* (DFS)

1. Pada algoritma DFS, pencarian *file* dilakukan secara rekursif.
2. Pertama-tama andaikan nama dari fungsinya ini adalah DFSMain, maka di awal fungsi DFSMain akan dilakukan penge-*list*-an untuk semua *file* dan *directory* pada *directory* awal di parameter DFSMain ini. Semua *file* dan *directory* yang berada disini menjadi sebuah *node child* untuk *directory* awal dari parameter DFSMain dan diberi warna hitam karena belum mengalami pengecekan.
3. Lalu dilakukan iterasi pada *file* dan *directory* yang ada di dalam *list*.
4. Lakukan iterasi terhadap *list directory* terlebih dahulu, pada iterasi ini akan dipanggil fungsi DFSMain dengan parameter *directory* awalnya merupakan *directory* yang ada di *list*. Lalu mengubah *node* dengan *directory* ini menjadi merah (dianggap gagal, jikalaupun ada nanti akan diubah ketika pengecekan *file*).
5. Lalu iterasi *list file* yang terdapat pada *directory* awal ini dan dicek dengan nama *file* yang ingin dicari, jika *file* sesuai maka *node* tersebut akan diubah ke warna biru beserta dengan *node parent* hingga *root*, dan jika tidak sesuai maka *node* akan menjadi warna merah. Apabila hanya ingin ditampilkan 1 *file* saja yang sesuai, maka program akan dilakukan break sehingga berhenti. Apabila ingin ditampilkan semua *file* yang sesuai, maka program akan berlanjut.
6. Basis dari algoritma rekursif ini adalah jika semua yang ada di dalam *directory* merupakan *file*. Dan rekurens jika dalam *directory* awal terdapat *directory* juga di dalamnya.

## 3.3 Ilustrasi Kasus

Misalkan jika ada sebuah *directory* dengan rincian *file* didalamnya sebagai berikut

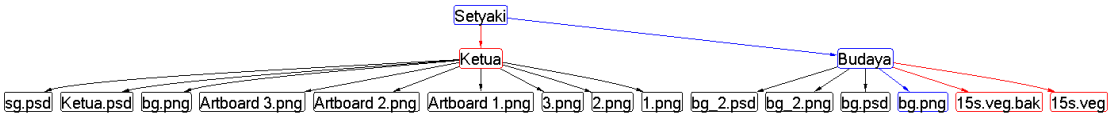



Dengan *directory* awalnya adalah Setyaki yang memiliki 2 *folder* di dalamnya yaitu Ketua dan Budaya. Dengan *file-file* yang berisi di dalam *folder* Ketua dan Budaya sesuai dengan *graf* di atas. Akan dilakukan pencarian terhadap *file* “bg.png” dengan algoritma BFS dan DFS dengan detail sebagai berikut.

### 3.2.1 Algoritma *Breadth First Search* (BFS)

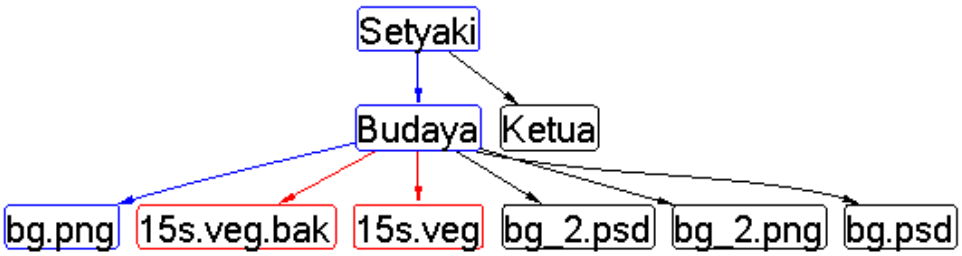
Dengan menggunakan agoritma BFS maka pertama-tama akan dilakukan *enqueue* untuk *folder*/*directory* Ketua dan Budaya. Sesuai dengan alfabetis, maka Budaya akan menjadi head. Lalu akan dilakukan *dequeue* sehingga pada *directory* Budaya yang merupakan *directory*/*folder* dan melakukan *enqueue* untuk setiap *file* dan *folder* di dalamnya. Lalu dilakukan *dequeue* dan mengembalikan *folder* "Ketua", oleh karena itu akan sama-sama dilakukan *enqueue* untuk setiap *file* dan *folder* yang ada di dalamnya.

Pada *dequeue* selanjutnya akan dicek *file* yang di *enqueue* dari *folder* Budaya yaitu dicek untuk *file* "15s.veg", "15.veg.bak", dan saat mengecek "bg.png" maka sudah sesuai dengan nama *file* yang diharapkan, maka program akan berhenti dan menjadikan *node* "bg.png" dan *parent-parent*nya menjadi biru. Sehingga terbentuk graf seperti berikut ini.




### 3.2.2 Algoritma *Depth First Search* (DFS)

Pada algoritma DFS, pertama kali yang dilakukan adalah memasukkan semua *file* dan *directory* pada *folder* awal Setyaki ke dalam *list*. Alhasil akan berisi *list folder* "Budaya" dan "Ketua" sehingga terbentuk sebagai *node child* dengan warna hitam. Selanjutnya dilakukan iterasi pada *list* tersebut dan ternyata saat pengecekan pertama adalah *folder* "Budaya" oleh karena itu program akan memanggil kembali fungsi DFS dengan *directory* awal "Budaya". Pada tahap ini akan dilakukan kembali memasukkan semua *file* dan *directory* ke dalam *list*, sehingga akan berisi 6 *file* yakni "15s.veg", "15s.veg.bak", "bg.png", "bg_2.psd", "bg_2.png", "bg.psd". Keenamnya merupakan *file* dan akan dibuat sebagai *node child* dengan warna hitam. Selanjutnya dilakukan pengecekan "15s.veg" dan "15s.veg.bak" yang bukan merupakan *file* yag dicari sehingga *node*nya berubah menjadi merah. Lalu saat pengecekan "bg.png" yang sudah sesuai dengan nama *file* yang dicari, program akan terhenti dan mengubah *node*nya menjadi warna biru. Sehingga terbentuk graf seperti di bawah ini.



# BAB 4

# IMPLEMENTASI DAN PENGUJIAN

## 4.1 Implementasi Program

User akan memilih metode BFS atau DFS di GUI. Bila user memilih metode BFS, maka algortima BFS akan dijalankan. Berikut Implementasi BFS di dalam file BFS.cs

```
function BFSMain(string dirPath, string searchFile,bool isAll, PictureBox pictureBox) -> Tuple of array
and Bitmap
# Algoritma
q = new queue()
edge = new edge()
ans = new ans()

# Masukkan dirPath ke queue
q.add(dirPath)

#Lakukan iterasi while hingga queue kosong
while(not(q.empty())) do
   string head = q.dequeue()

   # Kasus bila head adalah file
   if(isFile(head)) then
      if(isFile(head) and head==searchFile) then
         # masukan path dari head ke ans, buat graf, dan tandai jalur meunju jawaban dengan warna biru
         ans.add(head)
         bitmap = buildGraph(edge)
         pictureBox = bitmap
         markBlue(edge)

         # stop jika hanya diperlukan satu jawaban
         if(!isAll) then
            break
   # Kasus bila head adalah folder
   else
      # catat semua direktori
      pathsDir = GetDirectories(head)
      # catat semua file
      pathsFile = GetFiles(head)

      # masukkan semua pathsDir dan pathsFile ke queue dan edge
      q.enqueue(pathsDir)
      edge.add(pathDir)

      # build graf
      bitmap = buildGraph(edge)
      pictureBox = bitmap

   # Tandai warna merah untuk edge yang telah dikunjungi
   markRed(edge)
   # Tandai warna bitu untuk edge yang merupakan jawaban
   markBlue(edge)

# build graf
bitmap = buildGraph(edge)
pictureBox = bitmap

-> (ans,bitmap)
```

Bila User memilih metode DFS, algoritma DFS akan dijalankan. Berikut Implementasi DFS di dalam file DFS.cs

```
function DFS (string dirPath, string searchFile, bool isAll) -> array of string
    KAMUS
        ans : array of string
        edge : array of tuple <string, string, int>
        flag : boolean


    ALGORITMA
        ans <- new List<string>
        edge <- new List<Tuple<string, string, int>>
        flag <- 1

        DFSRecursive(dirPath, searchFile, isAll, pictureBox)


        i traversal [0..ans.Length]
            j traversal [0..edge.Count]
                # Mengecek jika ans[i] berada di dalam edge[j]
                if (isPrefixSubs(edge[j].Item1, ans[i]) and isPrefixSubs(edge[j].Item2, ans[i])) then
                    # Merubah semua edge file ditemukan dan parent-parentnya hingga root menjadi biru
                    edge[i] = Tuple.Create(edge[j].Item1, edge[j].Item2);
                    makeBlue(edge[i])
        # Mengembalikan list answer
        -> ans



procedure DFSRecursive(input string dirPath, string searchFile, bool isAll, input/output array of
string ans, array of tuple <string, string, int> edge, boolean flag)
    KAMUS
        pathsDir, pathsFile : array of string

    ALGORITMA
        # Mendapatkan semua directory dalam dirPath
        pathsDir <- Directory.GetDirectories(dirPath)
        # Mendapatkan semua file dalam dirPath
        pathsFile <- Directory.GetFiles(dirPath)


        # Masukkan pathsDir dan pathsFile ke dalam edge dan beri warna hitama
        i traversal [0..pathsDir.Length]
            makeBlack(edge.Add(Tuple.Create(dirPath, pathsDir[i])))
        i traversal [0..pathsFile.Length]
            makeBlack(edge.Add(Tuple.Create(dirPath, pathsFile[i])))

        # Kasus untuk folder
        i traversal [0..pathsDir.Length]
            # flag digunakan untuk menghentikan rekursif
            if (flag = true) then
                # Mengganti warna edge folder menjadi merah (dianggap tidak menemukan file yang sesuai)
                makeRed(edge.Add(Tuple.Create(dirPath, pathsDir[i])))
                DFSRecursive(path, searchFile, isAll, ans, edge, flag)
        # Kasus untuk file
        i traversal [0..pathsFile.Length]
            if (flag = true) then
                # Mengganti warna edge file menjadi merah (dianggap tidak menemukan file yang sesuai)
                makeRed(edge.Add(Tuple.Create(dirPath, pathsFile[i])))
                if (isFile(path)) then
                    # Cek apakah pathsFile[i] sama dengan searchFile
                    if (Path.GetFileName(path) = searchFile) then
                        ans.Add(path)
                        # isAll bernilai true artinya jika ingin didapatkan semua file yang sesuai
                        # isAll bernilai false artinya jika hanya ingin didapatkan 1 file saja yang
sesuai
```

```
                        if (!isAll) then
                            flag <- true
                            break
```

Untuk membuat graph nya, digunakan SearchingGraph.cs. Berikut adalah Implementasinya

```
function buildGraph(List of Tuple Egde,bool save) -> bitmap
# Algoritma
graph = new graph()

# traversal graf untuk memberi warna pada edge dan node
i traversal [1..Edge.nEdge]
   if(Edge.color=0) then
      if(Edge.node1.color!=RED and Edge.node1.color!=BLUE) then
         graph.addEgde(Edge.node1,Edge.node2,BLACK)
         Edge.node1.color = BLACK
         Edge.node2.color = BLACK
   else if(Edge.color=1) then
      if(Edge.node1.color=BLACK)
         graph.addEgde(Edge.node1,Edge.node2,RED)
         Edge.node1.color = RED
         Edge.node2.color = RED
   else
      graph.addEgde(Edge.node1,Edge.node2,BLUE)
      Edge.node1.color = BLUE
      Edge.node2.color = BLUE

# cari path yang memuat folder pictures jika bitmap perlu untuk disave
if(save)
   picture = GetCurrentDirectory()
   while(true) do
      # ambil seluruh folder di dalamnya
      pathsDir = GetDirectories()
      i traversal [1..pathsDir.n]
         if(getFolderName(pathsDir[i])="pictures") then
           # ditemukan lokasi folder pictures
           picture = pathsDir[i]
           break
      if(getFolderName(picture)="pictures") then
      break
   # simpan bitmap pada folder yang ditemukan
   save(bitmap,picture)
-> bitmap
```

### 4.2 Struktur Data yang Digunakan

Struktur data untuk *folder crawling* ini dibagi menjadi 4 program utama, Form1.cs, BFS.cs, DFS.cs, dan SearchingGraph.cs.

#### 4.2.1 Form1.cs

File ini mengatur *behaviour* untuk GUI yang telah dibuat, seperti tombol Search akan memanggil algoritma DFS atau BFS sesuai dengan metode yang dipilih, atau akan muncul path ke file yang ditemukan dan path tersebut bisa di tekan untuk membuka folder tempat file tersebut ada.

#### 4.2.2 BFS.cs

File ini berisi algoritma DFS untuk mencari file input dari path yang diinput juga. File ini menerima informasi dari Form1.cs dan hasilnya diberikan kepada SearchingGraph.cs untuk membuat graph nya. Hasil dari SearchingGraph.cs akan ditampilkan berulang-ulang sehingga menampilkan proses pencariannya.

### 4.2.1 DFS.cs

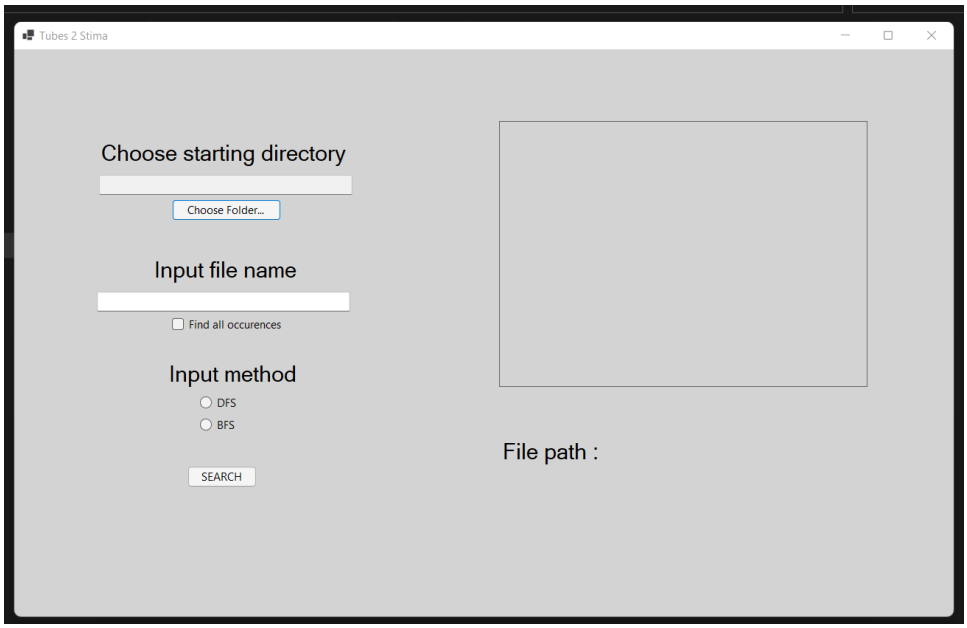
File ini berisi algoritma BFS untuk mencari file input dari path yang diinput juga. File ini menerima informasi dari Form1.cs dan hasilnya diberikan kepada SearchingGraph.cs untuk membuat graph nya. Hasil dari SearchingGraph.cs akan ditampilkan berulang-ulang sehingga menampilkan proses pencariannya.

### 4.2.1 SearchingGraph.cs

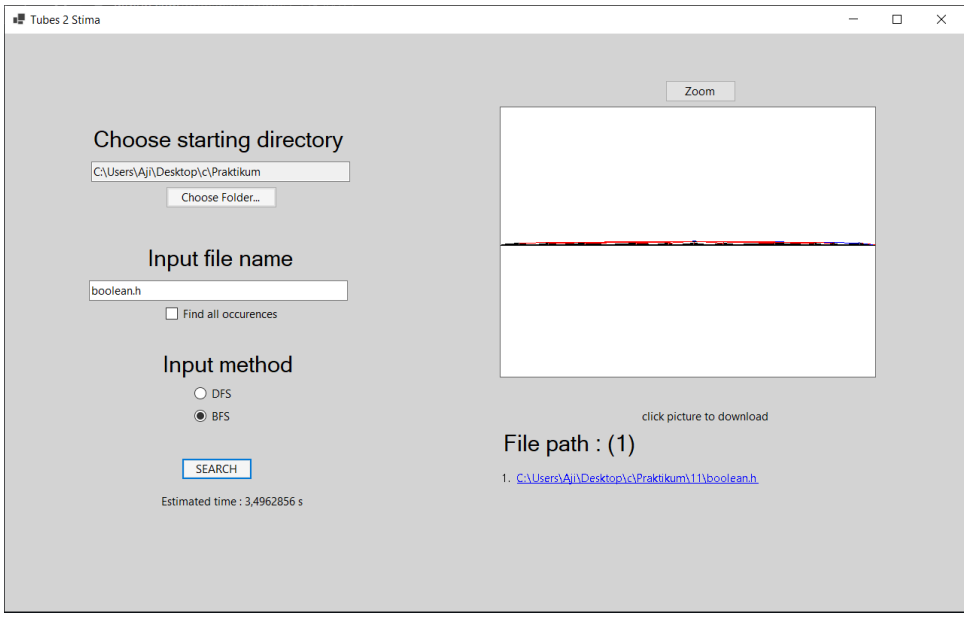
File ini berisi program yang akan mengolah hasil dari BFS.cs dan DFS.cs lalu mengubahnya menjadi sebuah graph yang diubah menjadi sebuah file .jpg. Di graph tersebut, jika berwarna hitam berarti belum dicari, jika berwarna biru berarti termasuk path ke file yang dicari, jika berwarna merah berarti path tersebut sudah sampai ujung tetapi tidak ditemukan filenya.

## 4.3 Tata Cara Penggunaan Program

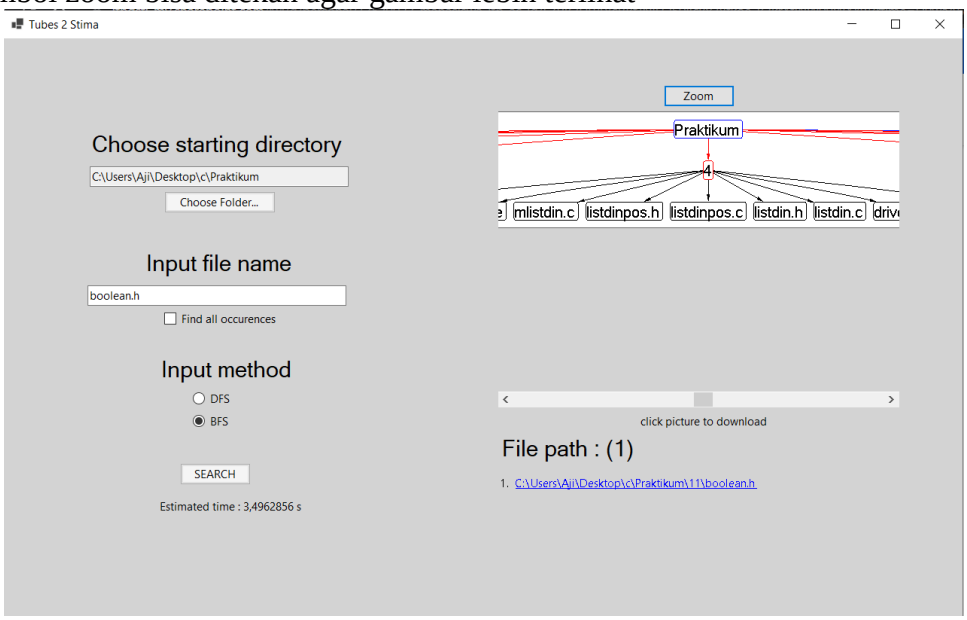
1. Melakukan clone pada repository ini (https:#github.com/primahafiz/Tubes2_13520012)
2. Masuk pada directory awal repository (Tubes2_13520012)
3. Jalankan program run.bat (klik 2x)
4. Aplikasi akan muncul seperti gambar dibawah ini :




5. Isi Form tersebut dan tekan tombol SEARCH
6. Tunggu proses algoritma selesai sambil melihat prosesnya
7. Setelah selesai, akan muncul seperti ini



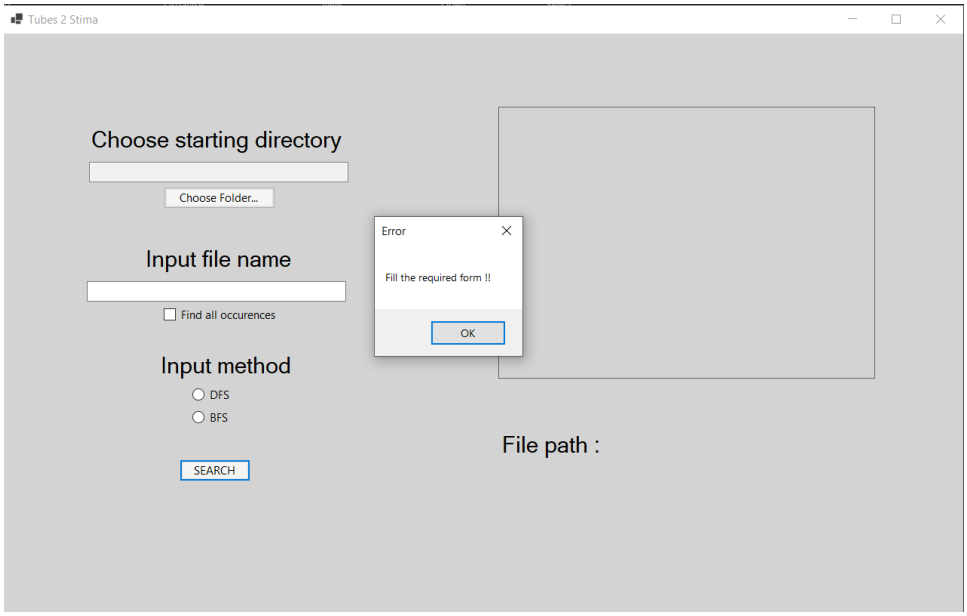

8. Tombol zoom bisa ditekan agar gambar lebih terlihat



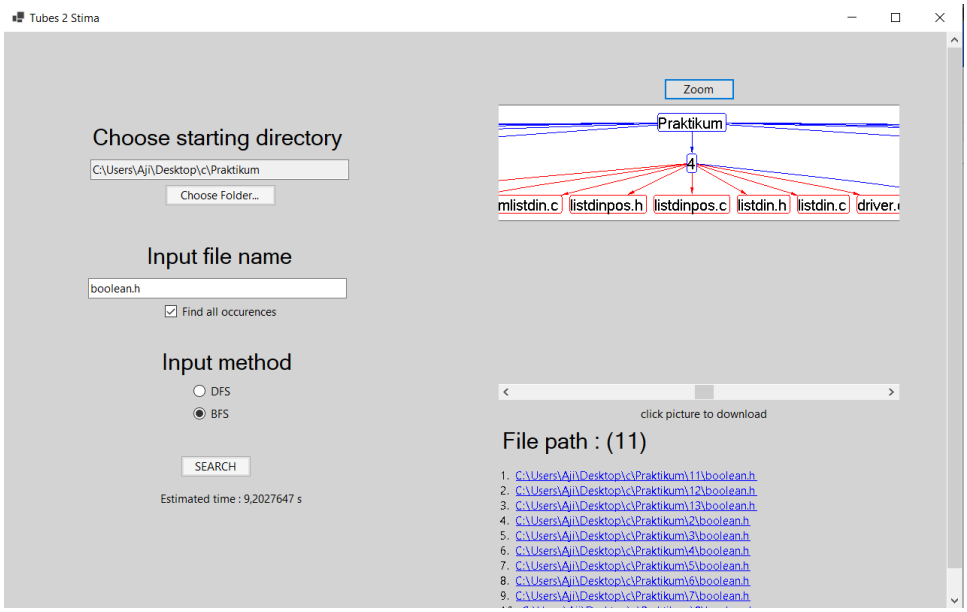

9. File path jika ditekan akan membuka file explorer ke file tersebut

### 4.4 Hasil Pengujian dan Analisisnya

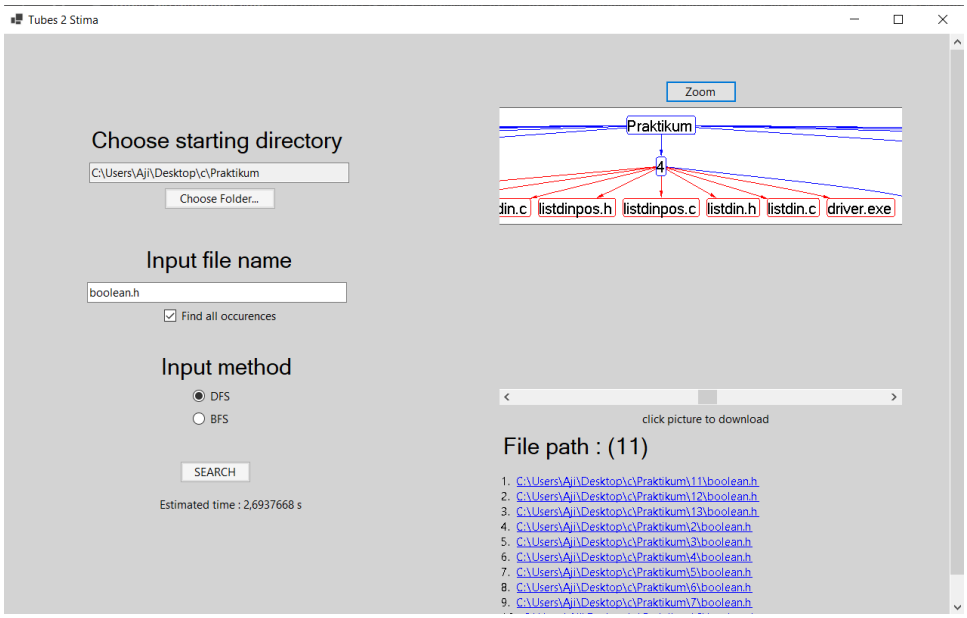
Jika ada bagian form yang belum diisi, akan dimunculkan sebuah error yang mengingatkan untuk mengisi form dengan penuh.




Setelah diuji, sebagian besar percobaan BFS akan menghasilkan waktu yang lebih lama dari DFS. Dibawah adalah hasil dari algoritma BFS.

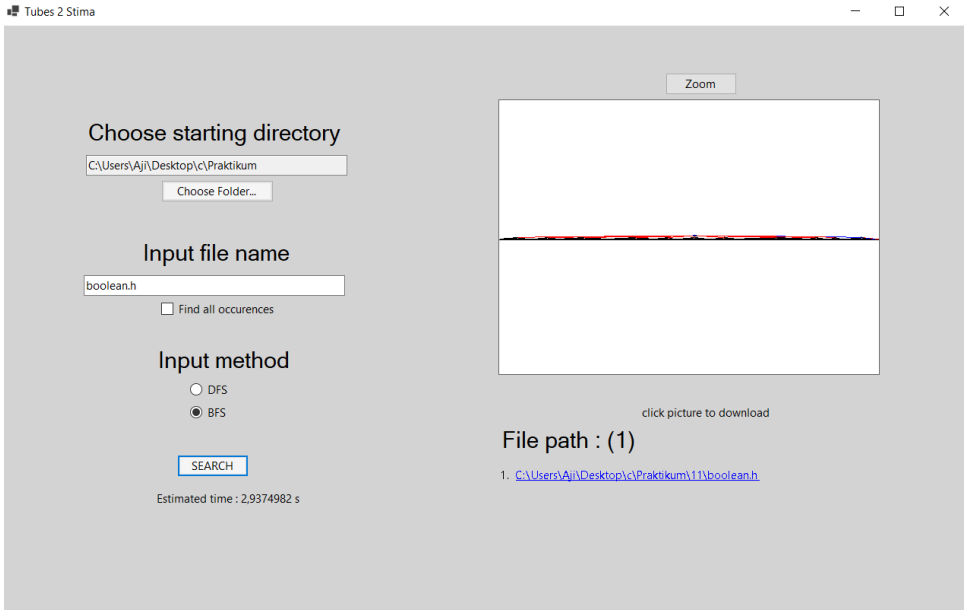


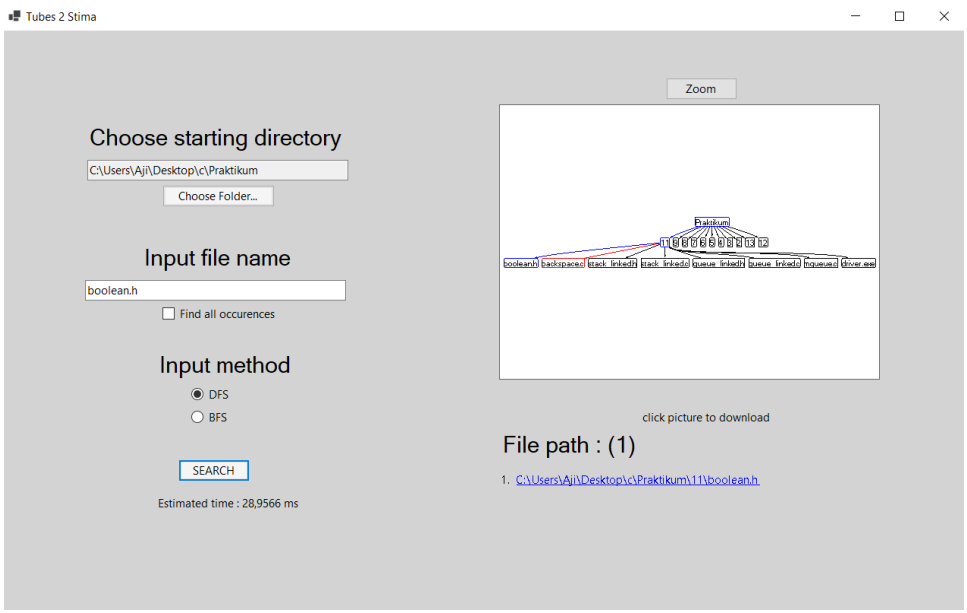
Bisa dilihat waktu untuk mencari semua kemungkinan yang dibutuhkan adalah sekitar 9.20 detik. Waktu yang ditampilkan adalah waktu dari algoritma ditambah waktu pembuatan graph, tidak murni algoritmanya saja. Sedangkan jika menggunakan DFS,




waktu yang dibutuhkan untuk mencari semua kemungkinan adalah sekitar 2.69 detik.

Jika yang dicari hanya yang muncul pertama saja, maka akan seperti ini.






BFS membutuhkan waktu hampir 3 detik sedangkan DFS hanya membutuhkan waktu 28.95 milidetik. Untuk pencarian yang hanya mencari kemungkinan pertama diatas, bisa dilihat algoritma DFS “hoki” menemukan file tersebut dibandingkan BFS. Akan tetapi untuk pencarian semua kemungkinan, BFS tetap lebih lama dibandingkan DFS. Hal ini disebabkan dari pembuatan graphnya. Di algoritma BFS, pembuatan graph lebih lama dikarenakan harus membuka foldernya satu-satu, sehingga file dan folder di graph akan lebih banyak yang di-generate dibandingkan DFS.

# BAB 5

# KESIMPULAN DAN SARAN

## 5.1 Kesimpulan dan Saran

Dalam menyelesaikan persoalan Folder Crawling ini dapat digunakan algoritma pencarian graf di antaranya adalah dengan metode Breadth First Search (BFS) dan Depth First Search (DFS). Dengan menggunakan metode BFS pencarian akan dilakukan secara melebar dan sesuai tingkatan kedalaman dari simpul sehingga simpul dengan tingkatan yang lebih rendah akan diperiksa lebih dahulu dibandingkan simpul dengan tingkatan yang lebih tinggi. Sedangkan pada DFS pencarian akan dilakukan secara mendalam sehingga ketika telah mencapai simpul daun akan dilakukan *backtracking*.

Pada persoalan ini setiap folder dan file dapat dianalogikan sebagai simpul pada graf dan dapat terhubung pada sebuah sisi. Dari hasil pengujian diperoleh bahwa pencarian dengan metode BFS dan DFS akan menghasilkan jawaban yang sama meskipun dengan waktu eksekusi yang berbeda. Waktu pencarian dengan metode BFS akan cenderung lebih lama dibandingkan DFS terutama jika lokasi file berada jauh dari *starting directory*.

# DAFTAR PUSTAKA

https:#informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

https:#codepolitan.com/pengenalan-bahasa-pemrograman-c-587effa1cb95b/

# LAMPIRAN

**Link Repository Github**
https://github.com/primahafiz/Tubes2_13520012

**Link Video Demo**
https://youtu.be/M92216hjO6Q