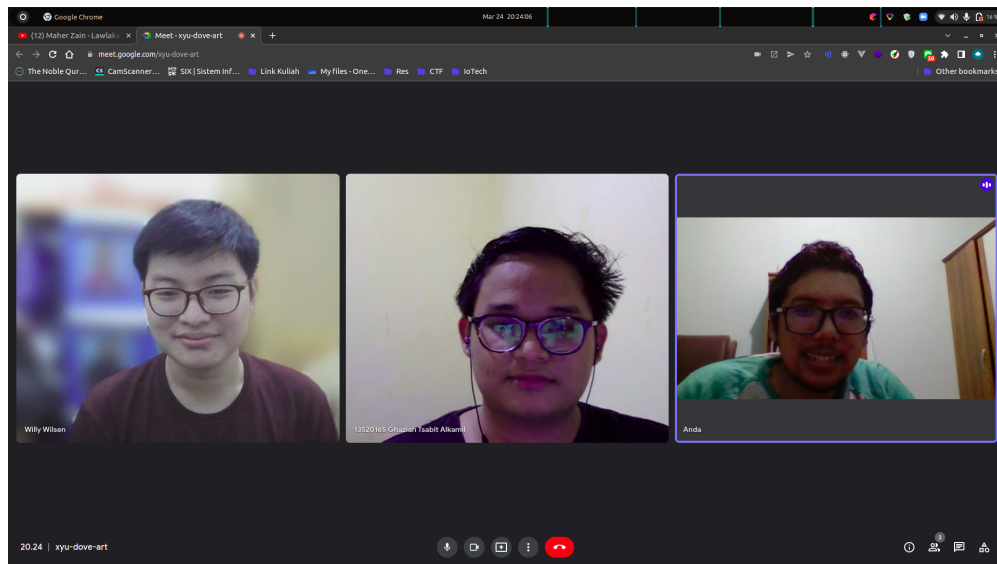


## **Laporan Tugas Besar 2** **IF2211 Strategi Algoritma**

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



Oleh :

13520128	Bayu Samudra
13520160	Willy Wilsen
13520165	Ghazian Tsabit Alkamil

**Sekolah Teknik Elektro dan Informatika**  
**Program Studi Teknik Informatika**  
**Institut Teknologi Bandung**  
**Tahun Ajaran 2021/2022**

## **DAFTAR ISI**

<b>BAB I</b>	<b>3</b>
<b>Deskripsi Tugas</b>	<b>3</b>
Deskripsi Tugas	3
Contoh Input dan Output Program	3
Spesifikasi Program	7
<b>BAB II</b>	<b>11</b>
<b>Landasan Teori</b>	<b>11</b>
<b>BAB III</b>	<b>18</b>
<b>Analisis Pemecahan Masalah</b>	<b>18</b>
Langkah langkah Pemecahan Masalah	18
Pada pembuatan aplikasi folder crawling, sebelumnya dilakukan analisis terlebih dahulu untuk memecahkan persoalan hingga menemukan file yang diinginkan. Langkah-langkah pemecahan persoalan yang didapatkan adalah sebagai berikut.	18
Proses Mapping Persoalan	18
Ilustrasi Kasus Lain	19
<b>BAB IV</b>	<b>21</b>
<b>Implementasi dan Pengujian</b>	<b>21</b>
Struktur Data Psedeocode	21
Psedeocode Program	22
Struktur Data Implementasi	26
List<T>	26
FileInfo	26
Queue<T>	26
Msagl.Graph	26
Stopwatch	26
Spesifikasi Program	27
Penggunaan Program	27
Hasil Pengujian	29
<b>BAB V</b>	<b>31</b>
<b>Kesimpulan dan Saran</b>	<b>31</b>
Kesimpulan	31
Saran	31
<b>DAFTAR PUSTAKA</b>	<b>32</b>

## **BAB I**

### **Deskripsi Tugas**

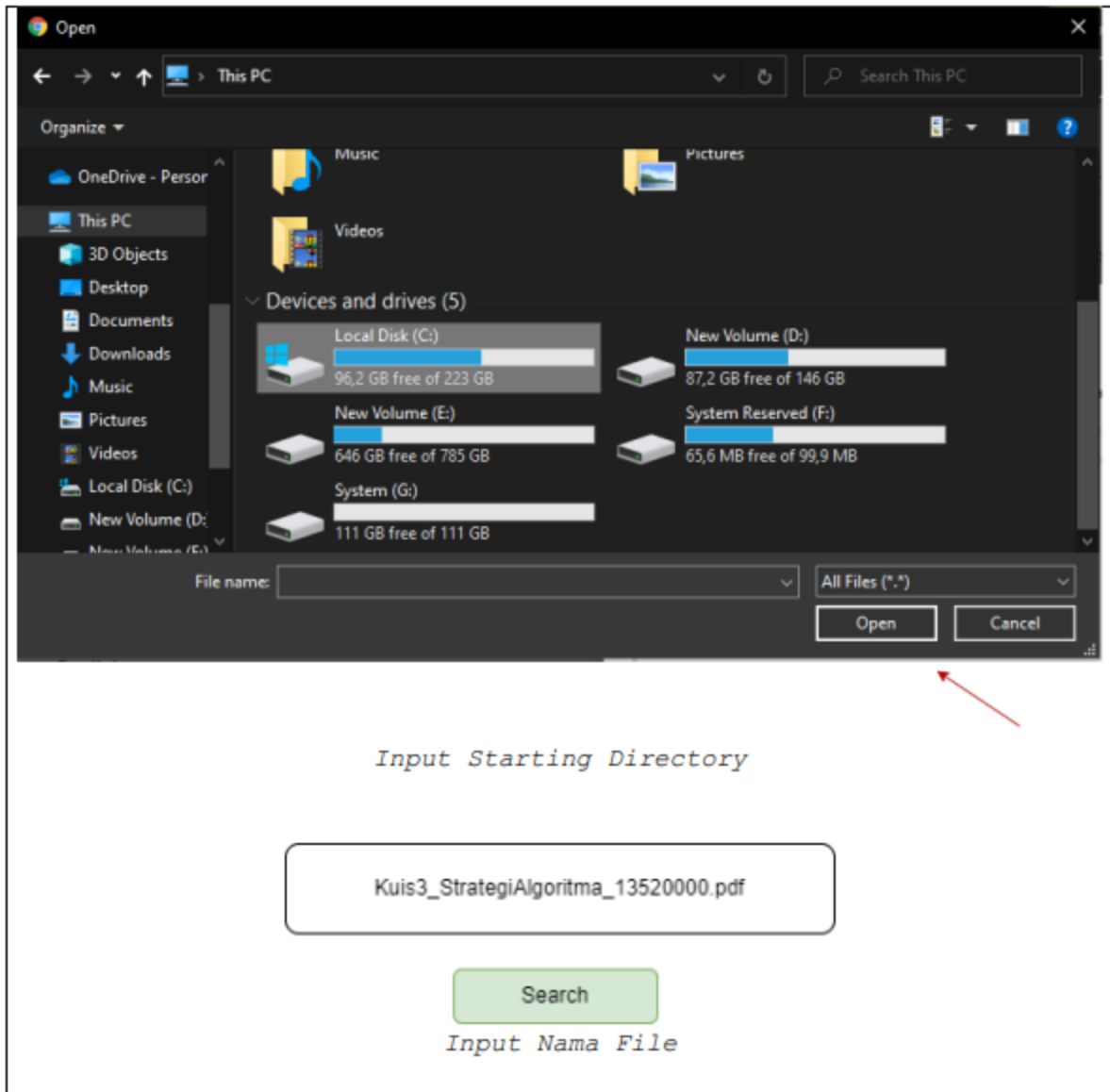
#### **A. Deskripsi Tugas**

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.

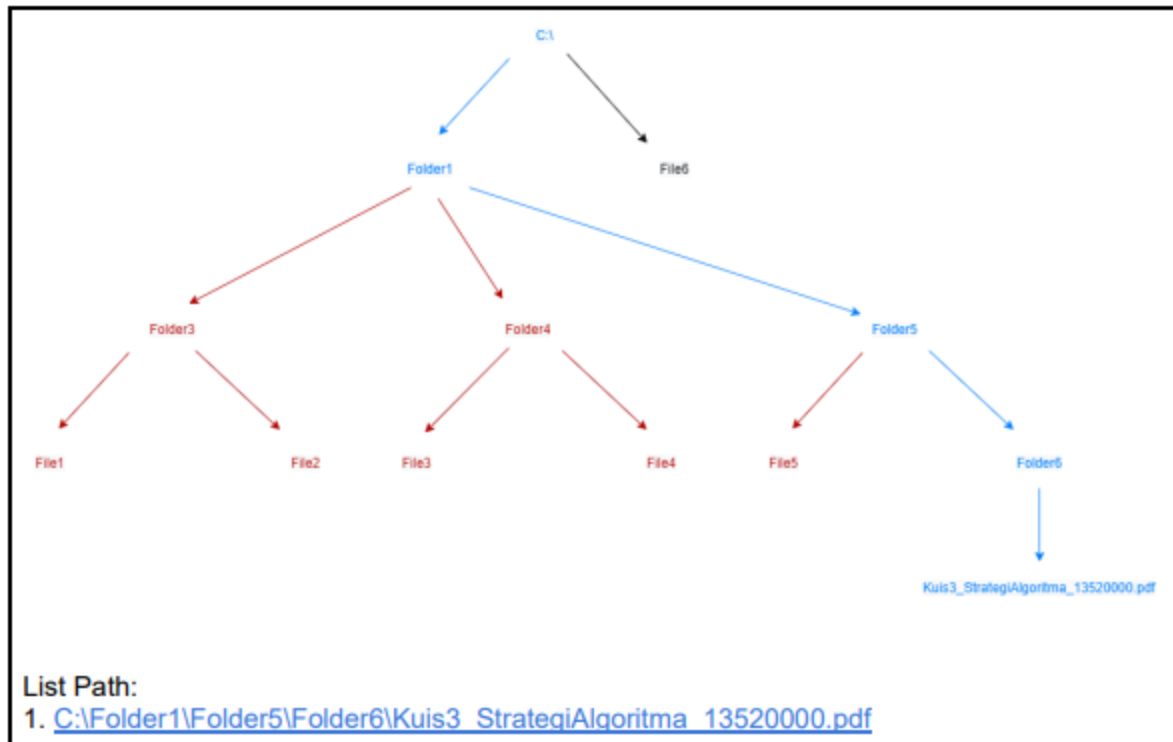
#### **B. Contoh Input dan Output Program**

Contoh masukan aplikasi:



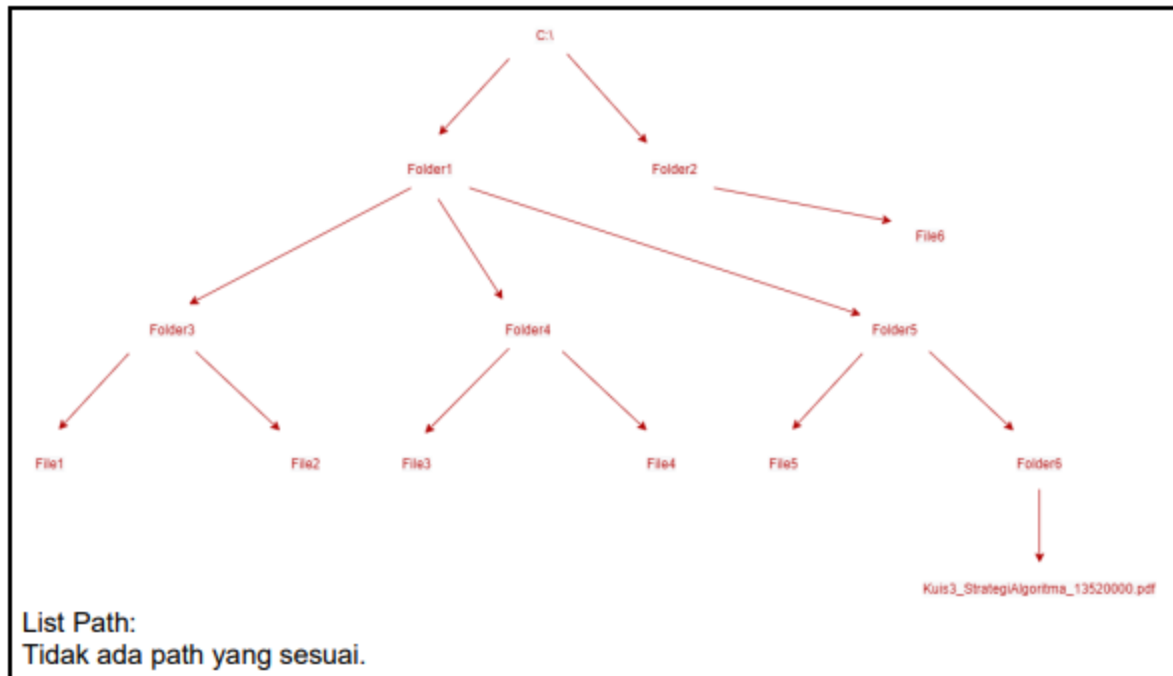
Gambar 1. Contoh Input Program

Contoh output aplikasi:



Gambar 2. Contoh output program

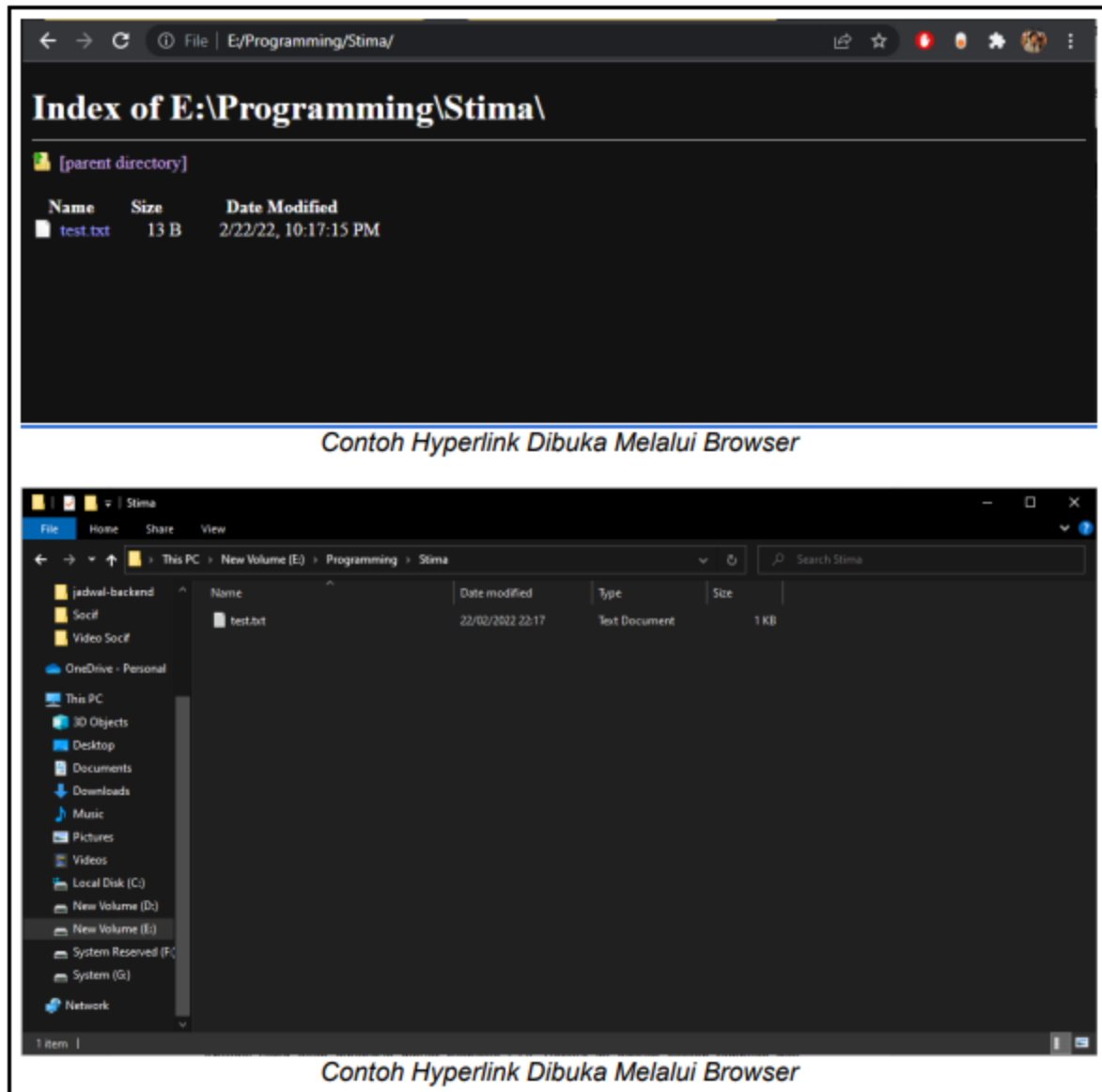
Misalnya pengguna ingin mengetahui langkah folder crawling untuk menemukan file Kuis3\_StrategiAlgoritma\_13520000.pdf. Maka, path pencarian DFS adalah sebagai berikut. C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → 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 2. Contoh output program jika file tidak ditemukan

Jika file yang ingin dicari pengguna tidak ada pada direktori file, misalnya saat pengguna mencari Kuis3Probststat.pdf, maka path pencarian DFS adalah sebagai berikut: C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3\_StrategiAlgoritma\_13520000.pdf → Folder6 → Folder5 → Folder1 → C:\ → Folder2 → File6. Pada gambar di atas, semua simpul dan cabang berwarna merah yang menandakan seluruh direktori sudah selesai diperiksa semua namun tidak ada yang mengarah ke tempat file berada.

Contoh Hyperlink pada path:



Gambar 4. Contoh ketika hyperlink di-klik

### C. Spesifikasi Program

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

### Folder Crawling

#### Input


Choose Starting Directory  
 No File Chosen

Input File Name

☐ Find all occurrence

Input Metode Pencarian  
☐ BFS  
☒ DFS

#### Output



### Folder Crawling

#### Input


Choose Starting Directory  
 C:/

Input File Name

☐ Find all occurrence

Input Metode Pencarian  
☐ BFS  
☒ DFS

#### Output



Path File :  
• [C:/Folder1/Folder5/Folder6/Kuis3\\_StrategiAlgoritma\\_13520000.pdf](#)

Time spent: 20.02s

Gambar 1. Tampilan layout dari aplikasi desktop yang dibangun

Spesifikasi GUI:

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



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

Program yang dibuat harus memenuhi spesifikasi wajib sebagai berikut:

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

Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk melakukan visualisasi adalah MSAGL (<https://github.com/microsoft/automatic-graph-layout>) Berikut ini adalah panduan singkat terkait penggunaan MSAGL oleh tim asisten yang dapat diakses pada: <https://docs.google.com/document/d/1XhFSpHU028Gaf7YxkmdbluLkQgVI3MY6gt1tPL30LA/edit?usp=sharing>

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

## BAB II

### Landasan Teori

#### A. Graf

Graf adalah struktur diskrit yang terdiri himpunan simpul (vertices, vertex) dan himpunan sisi (edges) yang menghubungkan simpul-simpul tersebut. Graf umumnya digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut.

Graf dapat didefinisikan sebagai berikut.

Graf  $G = (V, E)$ , yang dalam hal ini:

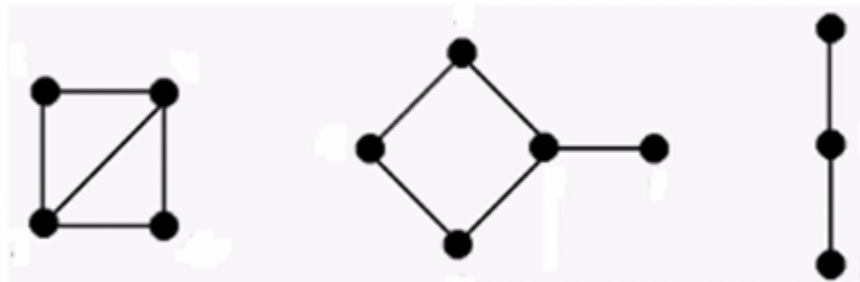
$V$  = Himpunan tidak-kosong dari simpul-simpul (vertices) =  $\{v_1, v_2, \dots, v_n\}$

$E$  = Himpunan sisi (edges) yang menghubungkan sepasang simpul =  $\{e_1, e_2, \dots, e_n\}$

Berdasarkan ada atau tidaknya sisi ganda pada suatu graf maka graf dibagi menjadi dua jenis:

##### 1. Graf sederhana

Graf sederhana adalah graf yang tidak mengandung gelang atau sisi ganda.



Gambar 2.a.1 Contoh Graf sederhana

(Sumber: <http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>)

##### 2. Graf tak sederhana

Graf tak sederhana dibedakan menjadi dua jenis:

###### 1. Graf ganda

Graf ganda adalah graf yang mengandung sisi ganda

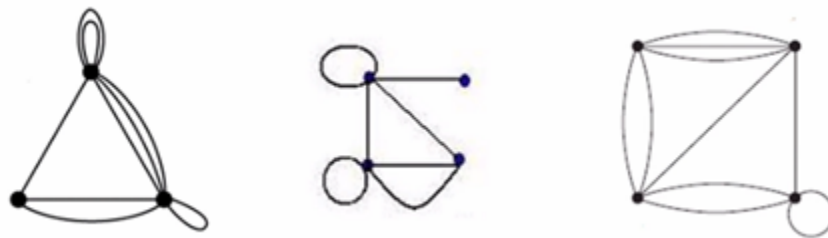


Gambar 2.a.2 Contoh Graf ganda

(Sumber: <http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>)

## 2. Graf semu

Graf semu adalah graf yang mengandung sisi gelang



Gambar 2.a.3 Contoh Graf semu

(Sumber: <http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>)

Berdasarkan orientasi arah pada sisi, graf dibagi menjadi dua jenis:

### 1. Graf tak-berarah

Graf tak-berarah adalah graf yang sisinya tidak mempunyai arah.

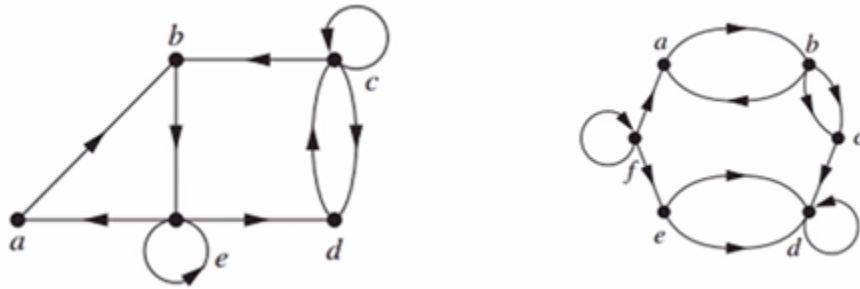


Gambar 2.a.4 Contoh Graf tak-berarah

(Sumber: <http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>)

## 2. Graf berarah

Graf berarah adalah graf yang setiap sisinya memiliki orientasi arah.



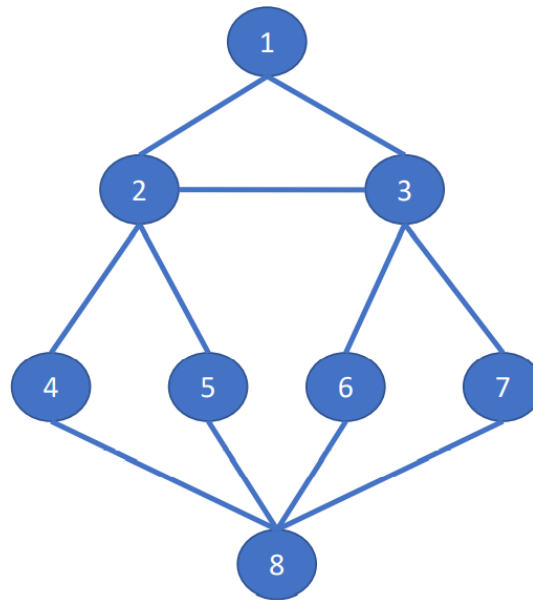
Gambar 2.a.5 Contoh Graf berarah

(Sumber: <http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>)

Berdasarkan proses pencarian solusi, graf dibagi menjadi dua pendekatan:

### 1. Graf statis

Graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dilakukan.



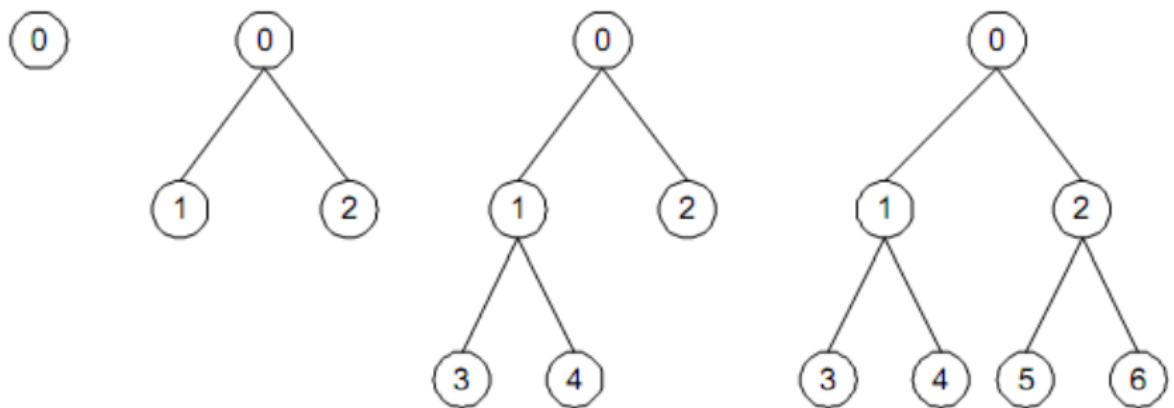
Gambar 2.a.6 Contoh Graf statis

(Sumber:

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Ba  
g1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Ba%20g1.pdf))

## 2. Graf dinamis

Graf dinamis adalah graf yang terbentuk saat proses pencarian dilakukan.



Gambar 2.a.7 Contoh Graf dinamis

(Sumber:

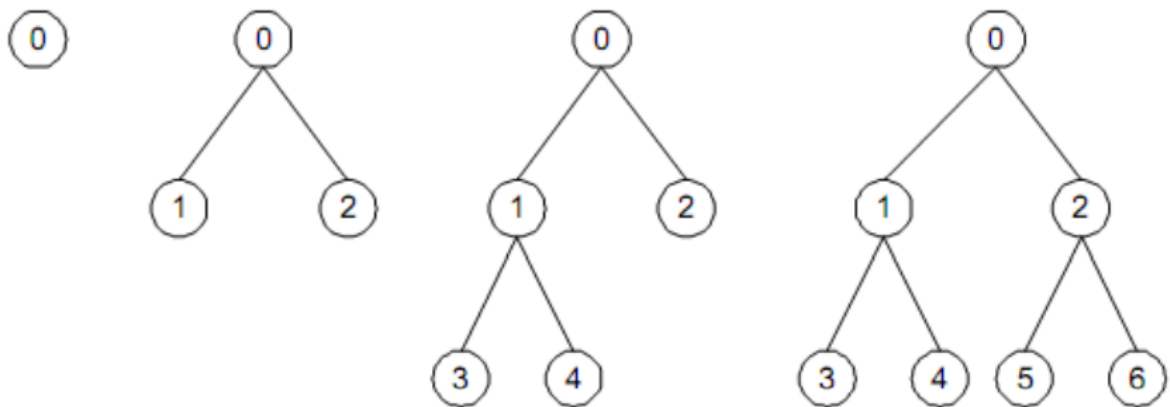
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Ba  
g2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Ba%20g2.pdf))

## B. BFS

BFS atau dikenal dengan *Breadth First Search* adalah proses pencarian yang dilakukan secara melebar. Dengan kata lain, BFS memprioritaskan pencarian per level. Misalkan traversal dimulai dari simpul  $v$ , maka algoritma untuk pencarian BFS adalah sebagai berikut.

1. Kunjungi simpul  $v$
2. Kunjungi semua simpul yang bertetangga dengan simpul  $v$  terlebih dahulu
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang sebelumnya sudah dikunjungi, demikian seterusnya.

Ilustrasi pembentukan pohon ruang status pada BFS dapat dilihat pada gambar berikut.



Gambar 2.b.1 Contoh Pembentukan pohon ruang status pada BFS

(Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Ba g2.pdf>)

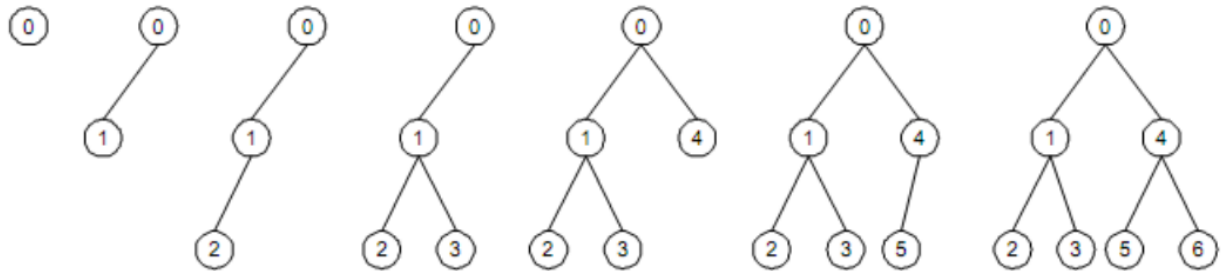
## C. DFS

DFS atau dikenal dengan *Depth First Search* adalah proses pencarian yang dilakukan secara mendalam. Dengan kata lain, DFS memprioritaskan pencarian secara rekursif. Misalkan traversal dimulai dari simpul  $v$ , maka algoritma untuk pencarian DFS adalah sebagai berikut.

1. Kunjungi simpul  $v$
2. Kunjungi simpul  $w$  yang bertetangga dengan simpul  $v$
3. Ulangi DFS mulai dari simpul  $w$
4. 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

5. Pencarian berakhir apabila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi

Ilustrasi pembentukan pohon ruang status pada BFS dapat dilihat pada gambar berikut.



Gambar 2.c.1 Contoh Pembentukan pohon ruang status pada DFS

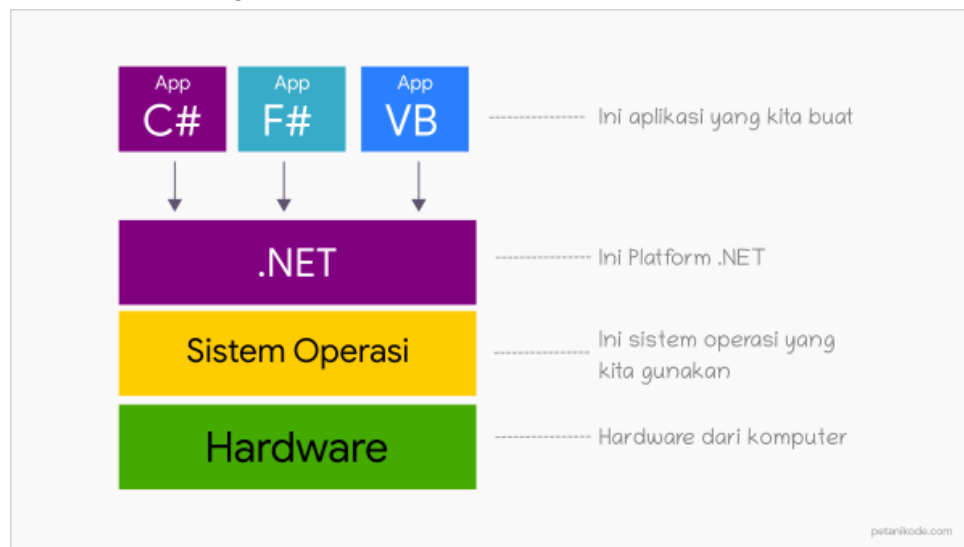
(Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Ba%20g2.pdf>)

#### D. Bahasa Pemrograman C#

C# adalah bahasa pemrograman yang dibuat oleh Microsoft dan ditargetkan berjalan di atas platform .NET.

.NET adalah mesin virtual yang digunakan untuk menjalankan program C#, F#, VB.NET dan program lainnya. Selain itu, .NET juga menyediakan *tools*, *library*, dan API yang dibutuhkan untuk membuat bahasa pemrograman C#. Ilustrasi hubungan antara C# dan .NET dapat dilihat pada gambar berikut.



Gambar 2.d.1 Hubungan antara C# dan .NET

(Sumber: <https://www.petanikode.com/cs-untuk-pemula/>)



Pada umumnya, program C# akan di-compile menjadi CIL (*Common Intermediate Language*) yang dipahami oleh .NET sehingga bahasa pemrograman C# cukup berbeda dengan bahasa pemrograman C dan C++ yang di-compile menjadi bahasa assembly dan dapat langsung dieksekusi oleh processor.

E. Desktop Application

Desktop Application atau Aplikasi Berbasis Desktop adalah suatu aplikasi atau software milik desktop (PC dan laptop) yang mampu beroperasi tanpa terhubung dengan koneksi internet (offline).

Aplikasi berbasis desktop dapat dibuat dengan menggunakan 3 bahasa pemrograman yaitu .NET, Java, dan Delphi. Bahasa pemrograman .NET dalam pembuatan aplikasi berbasis desktop meliputi Visual Basic (VB), C++, dan C# (C Sharp).

## BAB III

### Analisis Pemecahan Masalah

#### A. Langkah langkah Pemecahan Masalah

Pada pembuatan aplikasi *folder crawling*, sebelumnya dilakukan analisis terlebih dahulu untuk memecahkan persoalan hingga menemukan file yang diinginkan. Langkah-langkah pemecahan persoalan yang didapatkan adalah sebagai berikut.

1. Membentuk pohon statis pada file atau folder yang terdapat dalam *root directory* yang ingin diproses
2. Menambahkan seluruh file atau folder yang akan diproses ke dalam sebuah antrian bertipe list of string dengan mengikuti konsep algoritma BFS atau DFS
3. Memproses setiap elemen dari antrian tersebut dimulai dari elemen pertama
4. Apabila elemen yang sedang diproses merupakan file yang ingin dicari, maka pencarian berhasil
5. Apabila elemen yang sedang diproses bukan merupakan file yang ingin dicari, maka elemen tersebut dihapus dari antrian dan pemrosesan dilanjutkan ke elemen selanjutnya pada list. Demikian seterusnya hingga pencarian berhasil atau file tidak ditemukan sama sekali pada antrian tersebut

#### B. Proses Mapping Persoalan

Pada pembuatan aplikasi *folder crawling* perlu dilakukan proses mapping persoalan ke dalam elemen algoritma BFS dan DFS dalam bentuk representasi pohon statis.

Untuk implementasi algoritma BFS dalam aplikasi *folder crawling* maka proses mapping yang dilakukan adalah sebagai berikut:

1. Pohon ruang status : Pohon yang dibentuk selama pencarian BFS sedang dilakukan, ditandai dengan simpul atau daun yang diberi warna dengan urutan sesuai dengan aturan BFS yaitu membuat simpul atau daun yang berada pada level  $d$  terlebih dahulu, kemudian lanjut ke simpul yang berada pada level  $d + 1$ .
2. Simpul : Simpul-simpul yang dibuat berdasarkan folder dan file yang terdapat pada *root directory* yang ingin diproses
  - a. Akar : nama *root directory*
  - b. Daun : nama semua file yang ada di dalam directory yang ingin diproses
3. Cabang : operator *enqueue* ke dalam queue yang berisi simpul yang telah ditelusuri
4. Ruang status : seluruh simpul yang terdapat di dalam pohon ruang status algoritma BFS
5. Ruang solusi : himpunan dari semua jalur atau *path* yang menuju ke suatu daun, yang merupakan solusi adalah yang nama simpul daunnya sesuai dengan nama file yang ingin dicari

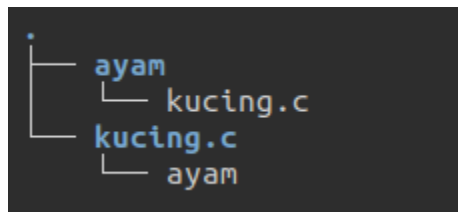
Untuk aplikasi algoritma DFS dalam aplikasi *folder crawling* maka proses mapping yang dilakukan adalah sebagai berikut:

1. Pohon ruang status : Pohon yang dibentuk selama pencarian DFS sedang dilakukan, ditandai dengan simpul atau daun yang diberi warna dengan urutan sesuai dengan aturan DFS yaitu melakukan penelusuran dari akar sampai daun, lalu melakukan penelusuran ke simpul atau daun lain apabila penelusuran sebelumnya sudah selesai
2. Simpul : Simpul-simpul yang dibuat berdasarkan folder dan file yang terdapat pada *root directory* yang ingin diproses
  - c. Akar : nama *root directory*
  - d. Daun : nama semua file yang ada di dalam directory yang ingin diproses
3. Cabang : operator *add* ke dalam list yang berisi simpul yang telah ditelusuri yang tersusun sesuai aturan DFS
4. Ruang status : seluruh simpul yang terdapat di dalam pohon ruang status algoritma DFS
5. Ruang solusi : himpunan dari semua jalur atau *path* yang menuju ke suatu daun, yang merupakan solusi adalah yang nama simpul daunnya sesuai dengan nama file yang ingin dicari

### C. Ilustrasi Kasus Lain

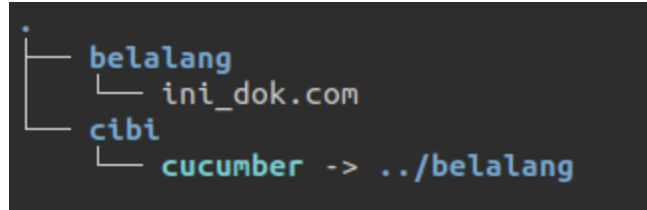
Beberapa contoh lain yang dapat digunakan sebagai kasus uji diantaranya adalah sebagai berikut:

- Penggunaan nama file yang sama dengan nama folder. Hal ini dapat digunakan untuk menguji apakah folder dan file dibedakan atau dibuat sama. Dengan menggunakan kasus uji ini juga dapat memastikan bahwa ilustrasi yang dibentuk merupakan tree. Berikut merupakan contoh instansiasi kasus ini:



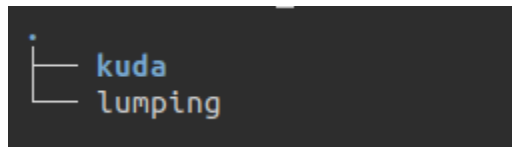
Gambar 3.c.1. Instansiasi kasus pertama

- Penggunaan file yang memiliki *symbolic link* terhadap file yang lain. Hal ini memastikan apakah pencarian tetap dapat dilakukan walaupun dalam keadaan *symbolic link*. Penggunaan kasus ini dapat berpotensi terjadi loop tidak terbatas dikarenakan memiliki link terhadap *parent*. Berikut ini adalah contoh instansiasi kasus ini:



Gambar 3.c.2. Instansiasi kasus kedua

- Folder yang tidak memiliki file sama sekali. Pada kasus ini, apakah terjadi eksepsi pada program pada saat terdapat folder yang tidak memiliki *child* sama sekali. Hal ini mencegah hal-hal yang tidak diinginkan karena adanya folder yang kosong. Berikut contoh instansi dari kasus ini:



Gambar 3.c.3. Instansiasi kasus ketiga

- File yang tersembunyi. Pada kasus ini, harus dipastikan semua file yang tersembunyi tetap bisa terjelajahi oleh program.

## BAB IV

### Implementasi dan Pengujian

#### A. Struktur Data Psedeocode

Misalkan terdapat fungsi-fungsi utilitas sebagai berikut:

- `getDirectories(path: string): string[]`  
Mengembalikan semua path *directory* pada path yang dimaksud. Bila path bukan merupakan folder, dikembalikan string kosong. Begitu pula bila tidak terdapat folder didalamnya.
- `getFiles(path: string): string[]`  
Mengembalikan semua path *file* pada path yang dimaksud. Bila path bukan merupakan folder, dikembalikan string kosong. Begitu pula bila tidak terdapat file didalamnya.
- `getName(path: string): string`  
Mengembalikan nama file/folder dari path

Berikut ini adalah Kelas Graf beserta definisi setiap methodnya

- `Graph(parent: Node): Graph`  
Membuat Objek Graf baru
- `Graph.addNode(parent: Node, node: Node): void`  
Membuat Node baru dengan parent merupakan parent node
- `Graph.addSubGraph(parent: Node, subGraph: Graph)`  
Menambahkan subgraph dengan memasan pada parent node
- `Graph.getChild(parent: Node): Node[]`  
Mengembalikan semua child parent
- `Graph.setStatus(node: Node, status: Status)`  
Membuat status dari node tersebut hingga ke induknya. Status bernilai beserta urutan prioritasnya adalah EXPANDABLE, NOT\_FOUND, FOUND. Bila induk memiliki prioritas yang lebih rendah, induk diset statusnya sesuai status terbarunya. Untuk setiap node baru, didefinisikan statusnya adalah EXPANDABLE.
- `Graph.showGraph()`  
Menampilkan graf dalam GUI

Berikut ini adalah Kelas Node beserta definisi setiap methodnya

- `Node(payload: any): Node`  
Membuat Node Graf baru
- `Node.getPayload(): any`  
Mendapatkan Node payload
- `Node.setStatus(status: Status)`  
Mengubah status suatu node

Berikut ini adalah definisi kelas queue

- `Queue()`: `Queue`  
Membuat objek queue
- `Queue.push(Item i)`  
Memasukan item di tail
- `Queue.pop()`: `item`  
Mengeluarkan item dari head
- `Queue.length()`; `int`  
Mengeluarkan jumlah item di queue

## B. Pseudocode Program

Berikut ini adalah implementasi penjelajah pencarian file menggunakan BFS dan DFS

```
procedure BFS(input path: string,
input fileName: string,
input isFindAll: boolean,
output g: Graph,
output foundPaths: string[])
KAMUS LOKAL
q: Queue
currentNode: Node
tmpNode: Node
directories, files: string[]
i: string
found: boolean

ALGORITMA
foundPaths := []
q := new Queue()
g := new Graph()
tmpNode := new Node(path)
q.push(tmpNode)

while q.length() > 0 do
    found := false
    currentNode := q.pop()
    directories := getDirectories(currentNode.getValue())
    files := getFiles(currentNode.getValue())

    if files.length() + directories.length() > 0 then
```

```
    for i in directories do
        tmpNode := new Node(i)
        q.push(tmpNode)
        g.addNode(currentNode, tmpNode)

    for i in files do
        if getName(i) == fileName then
            found := true

            tmpNode := new Node(i)
            g.addNode(currentNode, tmpNode)
            g.setStatus(tmpNode, FOUND)

            foundPaths.append(i)

            if not isFindAll then
                g.colorize()
                return
            else
                tmpNode := new Node(i)
                g.addNode(currentNode)
                g.setStatus(tmpNode, NOT_FOUND)
        else
            isFound := (getName(i) == fileName)
            if isFound then
                g.setStatus(currentNode, FOUND)
            else
                g.setStatus(currentNode, NOT_FOUND)

procedure DFS(input path: string,
    input fileName: string,
    input isFindAll: boolean,
    output g: Graph,
    output foundPaths: string[])
KAMUS LOKAL
    files, directories: string[]
    gchild: Graph
    tmp, fileNode: Node
    isFound: boolean
```

```
resultPath: string[]  
i: string
```

#### ALGORITMA

```
isFound := false  
tmp := new Node(path)  
g := new Graph(tmp)  
  
directories := getDirectories(path)  
files := getFiles(path)  
if directories.length() + files.length() > 0 then  
    for i in directories do  
        resultPath := []  
        DFS(i, fileName, isFindAll, gchild, resultPath)  
        g.addSubGraph(g, gchild)  
  
        if resultPath.length() > 0 then  
            foundPaths.insert(resultPath)  
            g.setStatus(tmp, FOUND)  
  
        if not isFindAll then  
            isFound := false  
            return  
  
    for i in files do  
        fileNode := new Node(i)  
  
        if getName(i) == filename then  
            isFound := true  
            foundPaths.append(i)  
  
            g.addNode(fileNode)  
            g.setStatus(fileNode, FOUND)  
  
        if not isFindAll then  
            return  
  
if not isFoundTmp then  
    g.setStatus(tmp, NOT_FOUND)
```



```
else
    isFound := (getName(i) == fileName)
    if isFound then
        tmp.setStatus(FOUND)
    else
        tmp.setStatus(NOT_FOUND)
```

Berikut ini adalah fungsi trigger yang dapat melakukan proses pencarian

```
procedure onClickSearch()
VARIABEL GLOBAL
    // Diisi menggunakan form dari GUI
    filename: string
    startPath: string
    mode: string
    isFindAll: boolean
    i: string
    start, finish, diff: Time

VARIABEL LOKAL
    g: Graph
    foundPaths: string[]

ALGORITMA
    start := now()
    if mode == BFS then
        BFS(startPath, filename, isFindAll, g, foundPaths)
    else
        DFS(startPath, filename, isFindAll, g, foundPaths)
    finish := now()

    diff := finish - start

    labelTime.text := diff.toFormattedTime()

    for i in foundPaths do
        listPathBox.append(i)
```

```
g.showGraph()
```

### C. Struktur Data Implementasi

Pada implementasi, terdapat struktur data yang digunakan, yaitu sebagai berikut

#### 1. List<T>

Struktur data list merupakan struktur data dinamis yang disediakan oleh .NET Framework. Struktur data ini merupakan *Linked List*. Beberapa method yang digunakan diantaranya adalah sebagai berikut:

- a. List<T>.Add(T data)  
Digunakan untuk menambahkan data pada list
- b. List<T>.ToArray()  
Mengembalikan array yang memiliki elemen dalam list
- c. List<T>.Clear()  
Menghapus semua elemen pada list
- d. List<T>.Count  
Mengembalikan nilai berupa jumlah elemen dalam list

#### 2. FileInfo

Tipe data ini digunakan untuk melihat atribut sebuah file. Hal tersebut bisa saja berupa nama file, nama folder induk, dan lainnya.

#### 3. Queue<T>

Queue merupakan struktur data yang menerapkan aturan *First In Last Out*. Tipe data ini telah disediakan oleh .NET Library sehingga tidak dilakukan implementasi secara langsung. Beberapa metode yang digunakan adalah sebagai berikut:

- a. Queue<T>.Enqueue(T item)  
Memasukan elemen dalam queue di belakang
- b. Queue<T>.Dequeue()  
Mengambil elemen pertama dan mengembalikannya

#### 4. Msagl.Graph

Objek ini digunakan untuk membentuk graf yang didukung langsung oleh MSAGL. Graf ini memrepresentasikan proses pencarian yang dilakukan menggunakan BFS atau DFS. Tipe data ini diupdate sepanjang proses pencarian.

#### 5. Stopwatch

Tipe data ini digunakan untuk melakukan perhitungan waktu eksekusi program.

## D. Spesifikasi Program

Untuk menjalankan program ini, diperlukan spesifikasi minimum sebagai berikut:

- Sistem Operasi : Microsoft Windows 10
- .NET Framework 4.7.2
- **(Optional)** Mono JIT Compiler 6.12.0.122

Untuk melakukan kompilasi program, diperlukan spesifikasi sebagai berikut

- **(Optional)** Visual Studio 2022
- **(Optional)** JetBrains Rider 2021.3.3

## E. Penggunaan Program

Program terdiri atas satu buah form yaitu sebagai berikut

Gambar 4.e.1. Gambar Form Program

Untuk memilih folder pencarian, dapat menekan tombol “Choose” pada bagian *Choose Starting Directory*. Nama file yang dicari diinputkan pada bagian *Input File Name*.

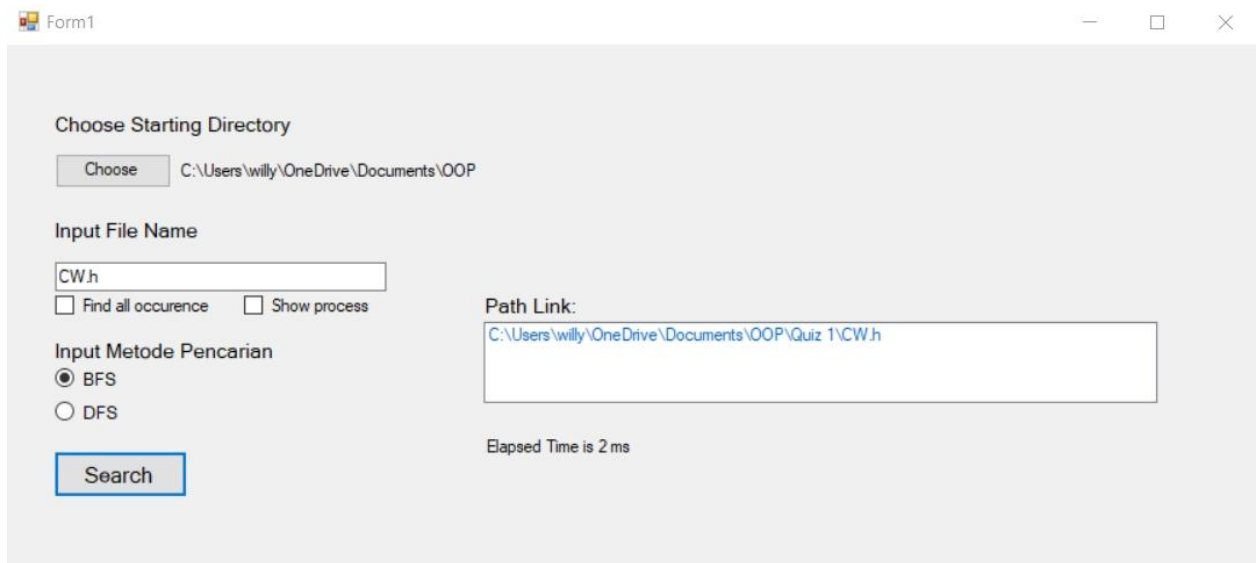
Terdapat dua opsi untuk melakukan pencarian yaitu

- *Find All Occurence*, yaitu mencari semua kemungkinan file pada folder yang diinputkan. Secara default, program hanya mencari hingga menemukan.
- *Show Process*, yaitu menampilkan proses pencarian file bertahap satu per satu

Terdapat dua buah mode pencarian, yaitu

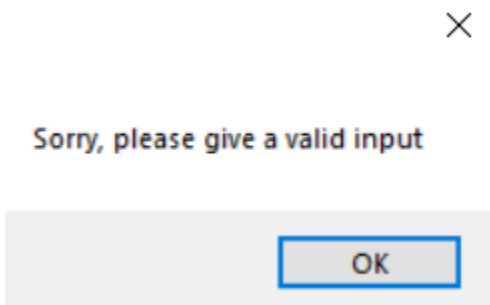
- BFS, yaitu mode pencarian dengan menggunakan algoritma BFS
- DFS, yaitu mode pencarian menggunakan algoritma DFS

Berikut ini adalah contoh pengisian yang benar.



Gambar 4.e.2. Gambar Form yang telah melakukan proses pencarian

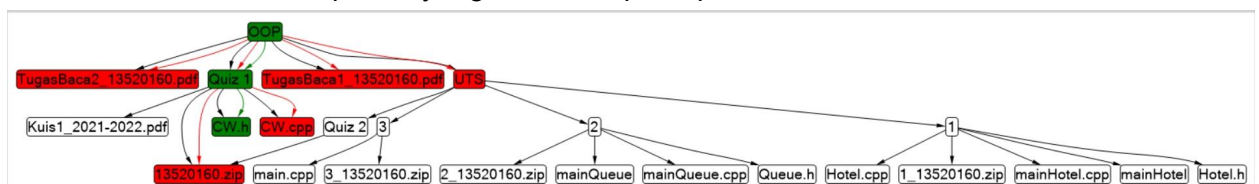
Jika input yang diberikan tidak valid, maka akan diberikan pesan kesalahan pada form.



Gambar 4.e.3 Gambar Form apabila tidak memberikan input yang valid

Bila ingin menjalankan program, tekan tombol search. Jika pencarian sudah selesai akan muncul form baru yang menggambarkan grafik yang dihasilkan. Apabila hasil tersebut ditutup, anda dapat menekan semua link yang didapatkan untuk membuka *file* tersebut.

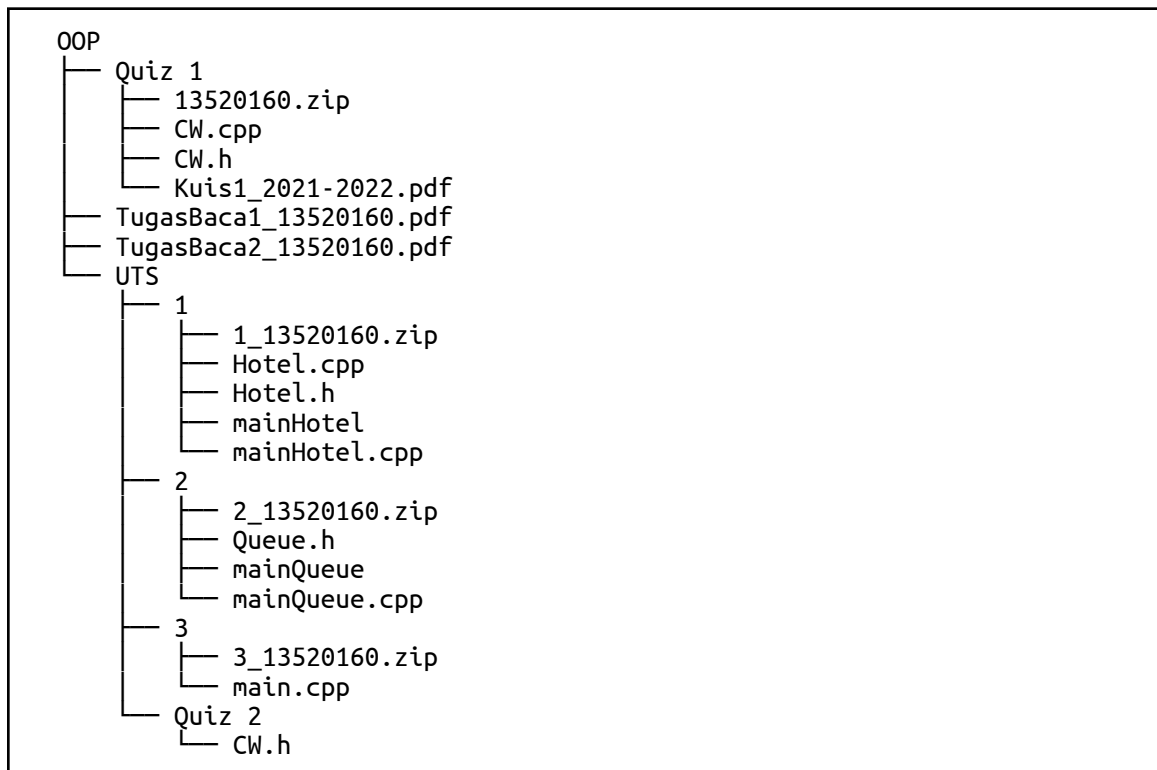
Berikut ini adalah contoh pohon yang terbentuk pada pencarian file.



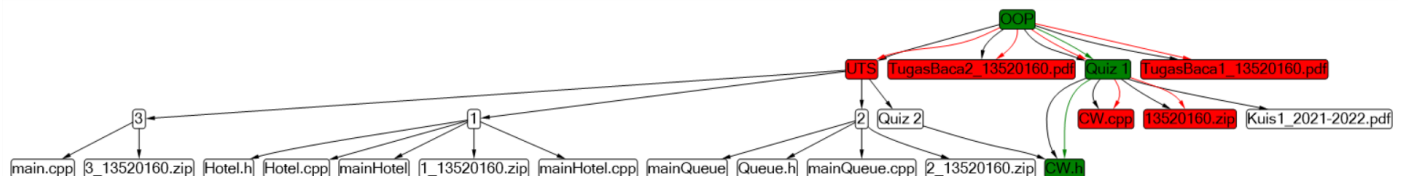
Gambar 4.e.4 Contoh pohon yang terbentuk pada pencarian file

## F. Hasil Pengujian

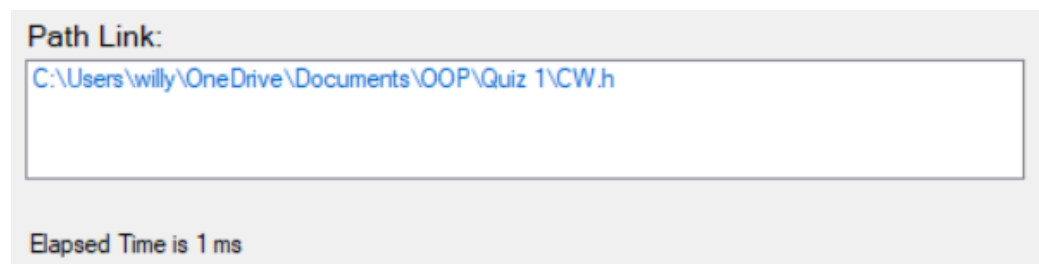
Struktur folder yang dipilih untuk proses pengujian adalah sebagai berikut.



### 1. Pencarian file CW.h pertama menggunakan BFS



Gambar 4.f.1 Pengujian pencarian pertama menggunakan BFS

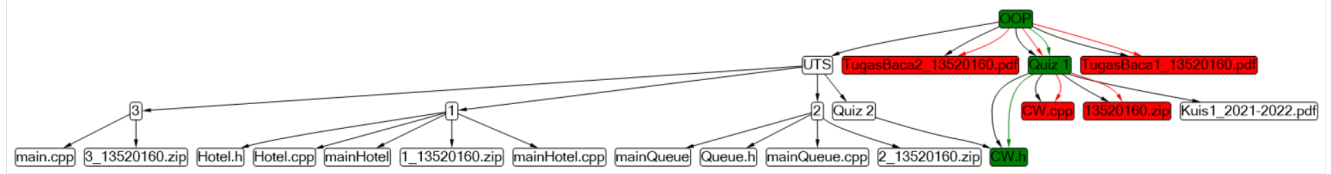


Gambar 4.f.2 Path dan estimasi waktu pencarian menggunakan BFS

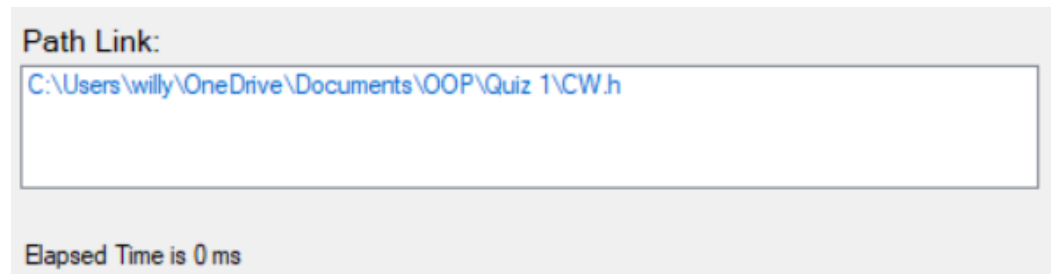
Pada gambar di atas, dapat dilihat bahwa seluruh file atau folder pada kedalaman 2 sudah diproses terlebih dahulu dengan urutan TugasBaca1\_13520160.pdf -> TugasBaca2\_13520160.pdf -> Quiz 1 -> UTS.

Pemrosesan kemudian dilanjutkan pada kedalaman 3 dimulai dari file atau folder yang terdapat dalam folder Quiz 1. Pada akhirnya, file CW.h pertama ditemukan pada path OOP -> Quiz 1 -> CW.h.

## 2. Pencarian file CW.h pertama menggunakan DFS



Gambar 4.f.3 Pengujian pencarian pertama menggunakan DFS



Gambar 4.f.4 Path dan estimasi waktu pencarian menggunakan DFS

Pada gambar di atas, pemrosesan dimulai dari TugasBaca1\_13520160.pdf. Tetapi karena TugasBaca1\_13520160.pdf merupakan file, maka proses diruntut-balik ke folder OOP kemudian melanjutkan kembali ke TugasBaca2\_13520160.pdf. Sama halnya dengan TugasBaca1\_13520160.pdf, maka proses berlangsung ke folder Quiz 1. Pada Quiz 1 dilakukan pembacaan seluruh file yang terdapat di dalamnya. Pada akhirnya, file CW.h pertama ditemukan pada path OOP -> Quiz 1 -> CW.h.

Dari kedua pemrosesan diatas, dapat dilihat bahwa pemrosesan DFS, Hal ini dikarenakan pemeriksaan yang dilakukan lebih sedikit dibandingkan dengan yang pada proses BFS. Dapat kita lihat juga pada kasus uji ini, file berada pada kedalaman 3. Bila menggunakan BFS, proses harus melakukan pemrosesan ke setiap simpul pada level 1 dan 2. Oleh karena itu, BFS akan banyak terjadi proses perbandingan.

## BAB V

### Kesimpulan dan Saran

#### A. Kesimpulan

Kesimpulan yang didapatkan dari hasil pengaplikasian algoritma BFS dan DFS dalam implementasi *folder crawling* adalah sebagai berikut:

1. Aplikasi folder crawling yang dihasilkan mampu mengimplementasikan algoritma BFS untuk melakukan pencarian file dengan baik.
2. Aplikasi folder crawling yang dihasilkan mampu mengimplementasikan algoritma DFS untuk pencarian file dengan baik
3. Pembentukan pohon berhasil diimplementasikan sesuai dengan spesifikasi yang ada.
4. Aplikasi folder crawling yang telah dibuat mampu menampilkan *path file* apabila file ditemukan pada saat proses pencarian.
5. Aplikasi folder crawling yang telah dibuat mampu melakukan pencarian semua kemunculan file pada folder yang ditelusuri.
6. Program dapat menampilkan progress pembentukan pohon dengan menambahkan node/simpul sesuai dengan pemeriksaan folder/file yang sedang berlangsung (Bonus berhasil diimplementasikan).
7. Aplikasi yang telah dibuat mampu menampilkan waktu eksekusi algoritma

Link repository github : [https://github.com/bayusamudra5502/Tubes2\\_13520128](https://github.com/bayusamudra5502/Tubes2_13520128)

Link video youtube : <https://youtu.be/nIDo4X6y2JU>

#### B. Saran

Saran yang dapat kelompok kami berikan yang berkaitan dengan pengaplikasian algoritma BFS dan DFS dalam implementasi folder crawling adalah sebagai berikut:

1. Pemahaman yang kuat tentang konsep algoritma BFS dan DFS sangat diperlukan dalam pembuatan aplikasi folder crawling ini agar dihasilkan algoritma pencarian yang efektif.
2. Diperlukan pemahaman yang cukup berkaitan dengan pemrograman berorientasi objek, karena pembuatan aplikasi folder crawling kali ini menggunakan bahasa pemrograman c#.
3. Perlu diperhatikan sistem operasi yang digunakan pada pembuatan aplikasi folder crawling, karena aplikasi folder crawling ini berbasis GUI.

## **DAFTAR PUSTAKA**

<http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>

<https://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>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Besar-2-IF2211-Strategi-Algoritma-2022.pdf>

<https://www.petanikode.com/cs-untuk-pemula/>

<https://github.com/microsoft/automatic-graph-layout>