

LAPORAN TUGAS BESAR 2

**IF2211/Strategi Algoritma
Semester II Tahun 2021/2022**



Dipersiapkan oleh:

Safiq Faray

13519145

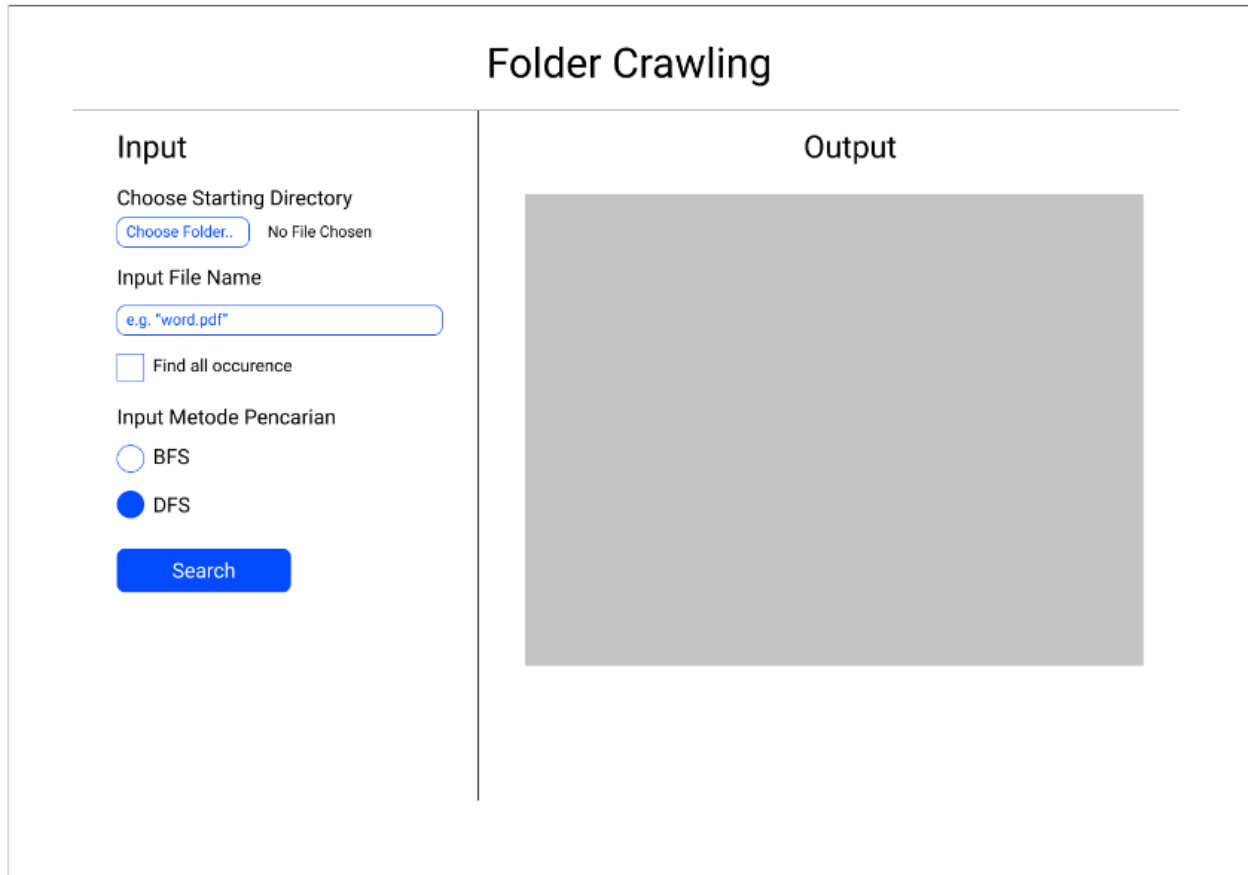
Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

I. Deskripsi Tugas

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

GUI yang akan dibangun.



The screenshot displays a desktop application window titled "Folder Crawling". The window is divided into two main sections: "Input" on the left and "Output" on the right. The "Input" section contains the following elements: a label "Choose Starting Directory" with a "Choose Folder.." button and the text "No File Chosen"; a label "Input File Name" with a text box containing "e.g. 'word.pdf'"; a checkbox labeled "Find all occurrence" which is currently unchecked; a label "Input Metode Pencarian" with two radio buttons, "BFS" (unselected) and "DFS" (selected); and a blue "Search" button. The "Output" section is a large, empty gray rectangular area.

Tampilan layout dari aplikasi desktop yang dibangun

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.

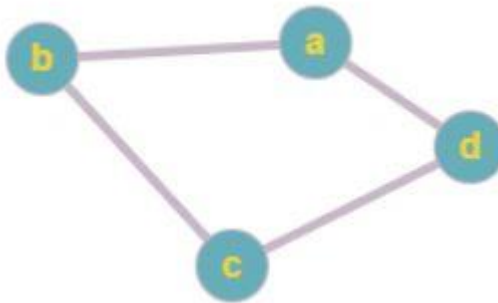
7. GUI dapat dibuat sekreatif mungkin asalkan memuat 5(6 jika mengerjakan bonus) spesifikasi di atas

II. Landasan Teori

A. Dasar Teori

Graph Traversal merupakan proses mengunjungi setiap *vertex/node*. *Graph Traversal* digunakan untuk mencari jalur dalam suatu *graph* dari titik asal ke titik tujuan, mencari jalur terpendek antara dua *node/vertex*, menemukan semua jalur yang bisa dilalui dari titik asal ke titik tujuan.

Berikut adalah contoh *graph* untuk proses *graph traversal*.



Struktur data *graph* tersebut dengan menggunakan dictionary adalah sebagai berikut:
`graph = {'a': ['b', 'd'], 'b': ['a', 'c'], 'c': ['b', 'd'], 'd': ['a', 'c']}`

Dictionary tersebut berisi semua *node/simpul* serta list tetangganya, dimana setiap key suatu *simpul* akan berkorespondensi dengan semua tetangganya yang terhubung dengan *simpul* key tersebut. Berikut ini adalah implementasi mencari jalur dari *simpul* asal ke *simpul* tujuan, jika ada jalur maka nilai yang dikembalikan adalah list path jika tidak ada jalur maka nilai yang dikembalikan *none*. Algoritma yang digunakan adalah *backtracking*, yaitu mencoba setiap kemungkinan secara bergantian sampai menemukan solusi.

Dalam Algoritma *traversal* graf sendiri terdapat dua pendekatan yaitu :

1. Pencarian melebar (*breadth first search/BFS*)

Breadth-first search adalah algoritma yang melakukan pencarian secara melebar yang mengunjungi *simpul* secara preorder yaitu mengunjungi suatu *simpul* kemudian mengunjungi semua *simpul* yang bertetangga dengan *simpul* tersebut terlebih dahulu. Selanjutnya, *simpul* yang belum dikunjungi dan bertetangga dengan *simpul-simpul* yang tadi dikunjungi, demikian seterusnya. Jika graf berbentuk pohon berakar, maka semua *simpul* pada aras *d* dikunjungi lebih dahulu sebelum *simpul-simpul* pada aras *d+1*.

Algoritma ini memerlukan sebuah antrian *q* untuk menyimpan *simpul* yang telah dikunjungi. *Simpul-simpul* ini diperlukan sebagai acuan untuk mengunjungi *simpul-simpul* yang bertetangga dengannya. Tiap *simpul* yang telah dikunjungi masuk ke dalam antrian hanya satu kali. Algoritma ini juga membutuhkan table

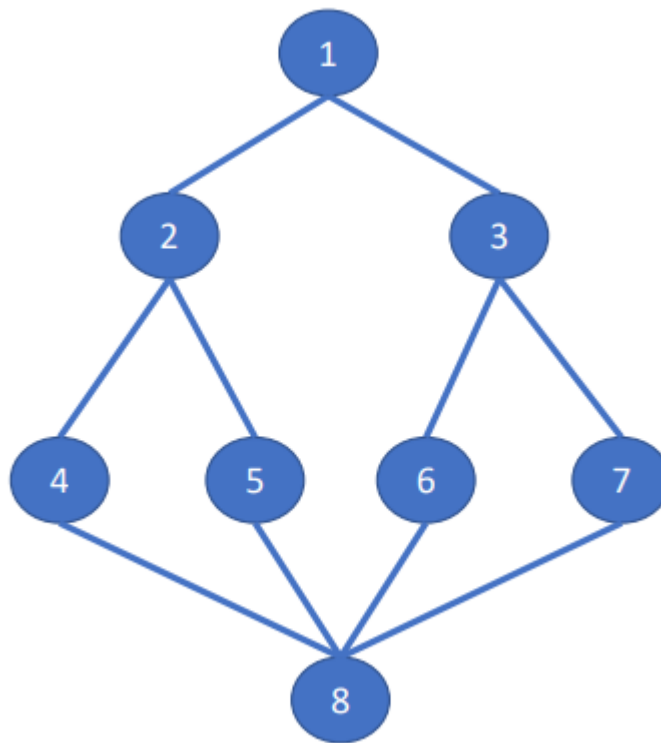
Boolean untuk menyimpan simpul yang telah dikunjungi sehingga tidak ada simpul yang dikunjungi lebih dari satu kali.

2. Pencarian mendalam (depth first search/DFS)

DFS (Depth-First-Search) adalah salah satu algoritma penelusuran struktur graf / pohon berdasarkan kedalaman. Simpul ditelusuri dari root kemudian ke salah satu simpul anaknya (misalnya prioritas penelusuran berdasarkan anak pertama [simpul sebelah kiri]), maka penelusuran dilakukan terus melalui simpul anak pertama dari simpul anak pertama level sebelumnya hingga mencapai level terdalam.

Setelah sampai di level terdalam, penelusuran akan kembali ke 1 level sebelumnya untuk menelusuri simpul anak kedua pada pohon biner [simpul sebelah kanan] lalu kembali ke langkah sebelumnya dengan menelusuri simpul anak pertama lagi sampai level terdalam dan seterusnya.

Jika keduanya dibandingkan dengan menyelesaikan 1 persoalan yang sama :



Dengan BFS urutan simpul yang dikunjungi : 1, 2, 3, 4, 5, 6, 7, 8

Dengan DFS urutan simpul yang dikunjungi : 1, 2, 4, 8, 5, 6, 3, 7

B. Penjelasan singkat mengenai C# desktop application development

Pada tugas kali ini, kami diberikan tugas untuk membuat sebuah aplikasi desktop. Banyak bahasa pemrograman diluar sana untuk membuat sebuah aplikasi desktop, salah satunya adalah C# desktop application development. Dengan menggunakan C# kita bisa membuat sebuah windows form yang terdiri atas textbox, button, label, dan masih banyak lagi.

Dari windows form yang dibuat kita bisa melakukan pemrograman terhadap control yang ingin berada pada GUI. Misalnya kita ingin membuat sebuah button yang menuliskan sebuah tulisan “Hello, world!”, kita bisa membuat button itu di form, lalu melakukan pemrograman terhadap button yang telah kita buat untuk menuliskan “Hello, world!” ke layar.

C# desktop application ini dapat diterapkan dengan menggunakan visual studio community yang dapat di download pada (<https://visualstudio.microsoft.com/vs/community/>).

III. Analisis Pemecahan Masalah

A. Langkah-langkah pemecahan masalah

Dalam tugas besar 2 Strategi Algoritma ini, kami disuruh untuk memetakan path dari folder/file ke dalam sebuah bentuk graph yang kemudian akan ditelusuri dengan cara BFS atau DFS.

Pengguna akan memasukkan folder awal yang akan ditelusuri, lalu setiap folder atau file didalamnya akan ditelusuri dan dicocokkan dengan nama file yang dicari oleh pengguna.

Pada folder yang telah dipilih, akan didata folder atau file anaknya. Jika dalam tahap itu telah ditemukan file yang dicari, maka program akan mengoutputkan gambar graf sesuai dengan telusurannya. Jika tidak, maka dari setiap folder yang ada, program akan mendata atau memasukkan data-data anakan dari folder tersebut menuju graph yang telah dibuat. Lalu, akan dilakukan pencarian lagi. Proses ini akan terus berjalan hingga file yang dicari telah ditemukan atau tidak ada folder lagi yang bisa ditelusuri.

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

Dalam program kami, kami membuat dua penyelesaian masalah seperti yang sudah diberikan pada spesifikasi tugas besar ini yaitu BFS dan DFS. BFS dan DFS yang kami buat sama-sama mengeluarkan output boolean yang memiliki parameter node mulai dan yang dituju. Kedua nya sama-sama menggunakan algoritma graph traversal, yang membedakan hanya jalurnya.

Pada algoritma BFS awalnya kami melakukan pemeriksaan pada node awal beserta dengan jalurnya. Semua itu diperiksa pada sebuah fungsi PointingTo yang memeriksa node menunjuk ke node mana saja. Setelah itu, dilakukan traversal ke semua node dan node yang sudah dikunjungi akan ditandai. Pada BFS ini kami menggunakan sistem antrian yang dimulai dari node awal, lalu berlanjut ke node-node yang ditunjuknya. Jika misal ada node yang ditunjuk tetapi node tersebut menunjuk ke node yang lain, maka node yang lain itu akan dimasukkan ke antrian.

Pada DFS hampir sama cara kerjanya dengan BFS, hanya saja pada DFS ini kami menggunakan rekursif. Pada DFS, jika ada node yang ditunjuk dan node tersebut menunjuk ke node yang lain maka node yang lain tersebut akan dimasukkan ke antrian. Jika BFS memasukkan langsung semua yang ditunjuk, DFS menunjuk hingga tidak ada yang bisa ditunjuk lagi. Kedua BFS dan DFS ini kami masukkan kedalam sebuah fungsi SearchPath yang memiliki parameter node awal, node akhir, dan BFS atau DFS. Pada SearchPath ini, kami memulai dari node yang dituju hingga ke node awal menggunakan jalur yang dipilih antara BFS atau DFS.

IV. Implementasi dan Pengujian

Implementasi Program

Program Graph

KAMUS

totalnode : integer

nodes : array [0..totalnode-1] of string

pointingTo : array [0..totalnode-1] of array [0..totalnode-1] of string

function findIdxInNodes(value : string) → integer

_____ { Mencari index dari nodes }

_____ { input adalah string node yg mau dicari }

_____ { output adalah index value dari list nodes }

procedure NewEdge(input thisNode : string, input toNode : string)

_____ { Menambahkan edge baru berbentuk array of string yang nantinya akan ditambah ke variable “pointingTo” }

function searchPath(from: string, target: string, with : string) → array [0..totalnode-1] of string

_____ { Mencari jalur terpendek dari node from ke target dengan BFS dan DFS }

_____ { Cara kerjanya adalah mencatat predecessor dari iterasi tiap node yang dicari oleh BFS/DFS, kemudian kita mencari path nya secara mundur, yaitu mulai target hingga from }

function BFS(from: string, target: string, pred: array [0..totalnode-1] of string) → boolean

_____ { Sebuah checker yang menghasilkan true jika dalam prediksi, goal dapat tercapai dalam menggunakan algoritma BFS }

function DFS(from: string, target: string, visit : array [0..totalnode-1] of boolean, pred: array [0..totalnode-1] of string, found : boolean) → boolean

_____ { Sebuah checker yang menghasilkan true jika dalam prediksi, goal dapat tercapai dalam menggunakan algoritma DFS }

ALGORITMA

function findIdxInNodes(value : string) → integer

_____ **KAMUS LOKAL**

i : integer

ALGORITMA

int i ← 0

while (i < totalnodes) do

{

if (value = pointingTo[i][0]) then

 → i

 }

 i ← i+1


```
}  
→ i
```

procedure NewEdge(input thisNode : string, input toNode : string)

KAMUS LOKAL

ALGORITMA

function searchPath(from: string, target: string, with : string) → array[0..totalnode-1] of string

KAMUS LOKAL

pred : array [0..totalnode-1] of string

path : array [0..totalnode-1] of string

currentNode : string

pathIdx : integer

found : boolean

visit : array [0..totalnode-1] of boolean

ALGORITMA

pathIdx ← 0

currentNode ← target

if (with = "BFS") then

if (BFS(from, target, pred)) then

 path[pathIdx] ← currentNode

 pathIdx ← pathIdx + 1

while (pred[findIdxInNodes(currentNode)] ≠ "#None#") do

 path[pathIdx] ← pred[findIdxInNodes(currentNode)]

 pathIdx ← pathIdx + 1

 currentNode ← pred[findIdxInNodes(currentNode)]

else if (with = "DFS") then

 i traversal [0..totalnode-1]

 pred[i] ← "#None#"

if (DFS(from, target, visit, pred, found)) then

 path[pathIdx] ← currentNode

 pathIdx ← pathIdx + 1

while (pred[findIdxInNodes(currentNode)] ≠ "#None#") do

 path[pathIdx] ← pred[findIdxInNodes(currentNode)]

 pathIdx ← pathIdx + 1

 currentNode ← pred[findIdxInNodes(currentNode)]

path.Reverse()

if (pathIdx = 0) then

 path[pathIdx] ← from

 pathIdx ← pathIdx + 1

 path[pathIdx] ← target

 pathIdx ← pathIdx + 1

 path[pathIdx] ← ""

 pathIdx ← pathIdx + 1

→ path

function BFS(from: string, target: string, pred: array [0..totalnode-1] of string) → boolean

KAMUS LOKAL ALGORITMA

function DFS(from: string, target: string, visit : array [0..totalnode-1] of boolean, pred: array [0..totalnode-1] of string, found : boolean) → boolean

KAMUS LOKAL ALGORITMA

```
visit[findIdxInNodes(from)] ← true
var traversal (pointingTo[findIdxInNodes(from)])
  if (!visit[findIdxInNodes(val)]) then
    pred[findIdxInNodes(val)] ← from
    DFS(val, target, visit, pred, ref found)
  if (val = target) then
    found ← true
```











→ found

Penjelasan struktur data yang digunakan dalam program dan spesifikasi program

Pada file Graph.cs, terdapat class utama yang menjadi graph yang merepresentasikan folder/file yang ditelusuri. Program yang mengatur GUI terdapat pada Form1.cs

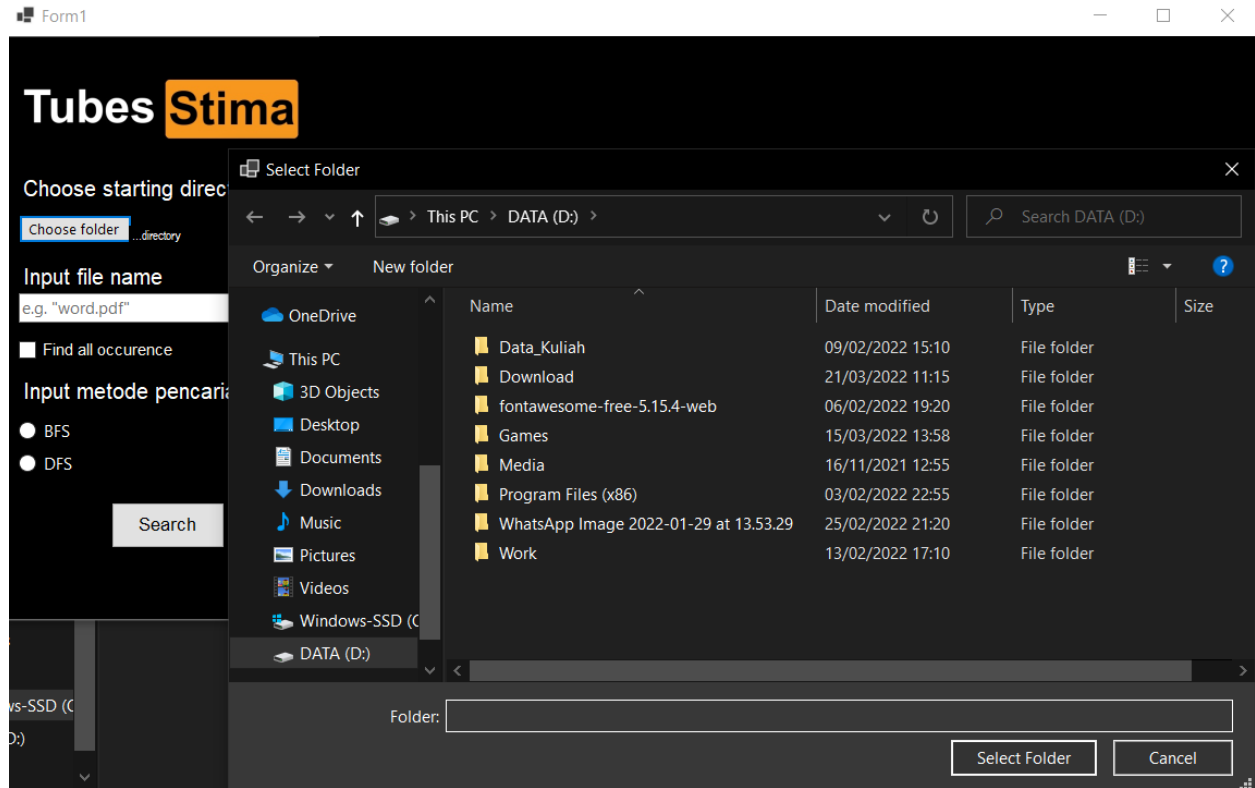
Penjelasan tata cara penggunaan program

1. Buka file Bin, klik 2 kali pada TubesStima2.exe

Name	Date modified	Type	Size
 AutomaticGraphLayout.dll	12/08/2020 14:07	Application extens...	1.565 KB
 AutomaticGraphLayout.Drawing.dll	16/11/2020 0:51	Application extens...	148 KB
 graph.png	26/03/2022 1:46	PNG File	1 KB
 Microsoft.Msagl.GraphViewerGdi.dll	16/11/2020 1:28	Application extens...	153 KB
 Microsoft.Msagl.WpfGraphControl.dll	16/11/2020 0:52	Application extens...	66 KB
 TubesStima2.deps.json	25/03/2022 21:40	JSON Source File	4 KB
 TubesStima2.dll	26/03/2022 2:08	Application extens...	129 KB
 TubesStima2.exe	26/03/2022 2:08	Application	146 KB
 TubesStima2.pdb	26/03/2022 2:08	Program Debug D...	19 KB
 TubesStima2.runtimeconfig.json	25/03/2022 21:40	JSON Source File	1 KB

2. Program akan muncul. Pencet browse untuk memasukkan folder

3. Pencet refresh untuk memunculkan, pilih akun dan teman yang ingin di explore



4. Pilih algoritma BFS maupun DFS, cari nama file, lalu klik "Search"

Form1

Tubes Stima

Choose starting directory

Choose folder

Input file name

☐ Find all occurrence

Input metode pencarian

☐ BFS

☒ DFS

D:\Media\Texts\Desain

D:\Media\Texts\Desain\geemu.docx

[D:\Media\Texts\Desain\geemu.docx](#)

Hasil Pengujian

Test 1

Form1

Tubes Stima

Choose starting directory

Choose folder

Input file name

☐ Find all occurrence

Input metode pencarian

☐ BFS

☒ DFS

D:\Data_Kuliah\test1

D:\Data_Kuliah\test1\test2.txt D:\Data_Kuliah\test1\folder D:\Data_Kuliah\test1\folder

D:\Data_Kuliah\test1\folder\test1.txt D:\Data_Kuliah\test1\folder\test1 - Copy.txt D:\Data_Kuliah\test1\folder\test1 - Copy(2).txt

[D:\Data_Kuliah\test1\folder\test1.txt](#)

Test 2

Form1

Tubes **Stima**

Choose starting directory

Choose folder D:\Data_Kuliah\test2

Input file name

test3.txt

☐ Find all occurrence

Input metode pencarian

☒ BFS

☐ DFS

Search

D:\Data_Kuliah\test2

D:\Data_Kuliah\test2\test4.txt

D:\Data_Kuliah\test2\test3.txt

D:\Data_Kuliah\test2\folder2 - Copy

D:\Data_Kuliah\test2\folder2

D:\Data_Kuliah\test2\test3.txt

Analisis dari desain solusi algoritma BFS dan DFS yang diimplementasikan

```
1 reference
public Boolean BFS(string from, string target, ref string[] pred, ref string targetpath)
{
    string first; //untuk traversal saja, ignore this
    bool[] visit = new bool[totalnode]; //menandai apakah node-node telah dikunjungi atau tidak
    List<string> list;
    for (int i = 0; i < totalnode; i++)
    {
        visit[i] = false; //default value : false
        pred[i] = "#None#";
    }

    // antrian node untuk dikunjungi
    List<string> queue = new List<string>();

    // Node pertama
    visit[findIdxInNodes(from)] = true;
    queue.Add(from);

    while (queue.Count != 0)
    {
        // Dequeue, masukkan ke path
        first = queue.First();
        queue.RemoveAt(0);

        // Melist pointingTo agar diproses ; in short mencari tetangga dari node tsb.
        list = pointingTo[findIdxInNodes(first)];

        foreach (var val in list)
        {
            if (!visit[findIdxInNodes(val)])
            {
                visit[findIdxInNodes(val)] = true;
                pred[findIdxInNodes(val)] = first;
                queue.Add(val);

                if (Path.GetFileName(val).Equals(target))
                { //Jika ketemu, maka akan return true
                    targetpath = val;
                    return true;
                }
            }
        }
    }
}
```

Seperti yang bisa dilihat pada program diatas pada algoritma BFS awalnya kami melakukan pemeriksaan pada node awal beserta dengan jalurnya. Semua itu diperiksa pada sebuah fungsi PointingTo yang memeriksa node menunjuk ke node mana saja. Setelah itu, dilakukan traversal ke semua node dan node yang sudah dikunjungi akan ditandai. Pada BFS ini kami menggunakan sistem antrian yang dimulai dari node awal, lalu berlanjut ke node-node yang ditunjuknya. Jika misal ada node yang ditunjuk tetapi node tersebut menunjuk ke node yang lain, maka node yang lain itu akan dimasukkan ke antrian.

```

2 references
public Boolean DFS(string from, string target, Boolean[] visit, string[] pred, ref Boolean found, ref string targetpath)
{
    //Penerapan algoritma BFS
    //Cara kerja (dalam konteks pencarian path) hampir sama dengan BFS.
    visit[findIdxInNodes(from)] = true; //ditandai bahwa node from sudah dikunjungi
    foreach (var val in pointingTo[findIdxInNodes(from)]) //traversal adjascent node
    {
        if (!visit[findIdxInNodes(val)]) //jika belum dikunjungi, maka diproses
        {
            pred[findIdxInNodes(val)] = from; //dicatat predecessornya
            DFS(val, target, visit, pred, ref found, ref targetpath); //rekursif
            if (Path.GetFileName(val).Equals(target))
            {
                targetpath = val;
                found = true; //jika ketemu, maka mengganti variable found menjadi true
            }
        }
    }
    return found;
}
}

```

Pada DFS hampir sama cara kerjanya dengan BFS, hanya saja pada DFS ini kami menggunakan rekursif. Pada DFS, jika ada node yang ditunjuk dan node tersebut menunjuk ke node yang lain maka node yang lain tersebut akan dimasukkan ke antrian. Jika BFS memasukkan langsung semua yang ditunjuk, DFS menunjuk hingga tidak ada yang bisa ditunjuk lagi. Kedua BFS dan DFS ini kami masukkan kedalam sebuah fungsi SearchPath yang memiliki parameter node awal, node akhir, dan BFS atau DFS. Pada SearchPath ini, kami memulai dari node yang dituju hingga ke node awal menggunakan jalur yang dipilih antara BFS atau DFS.

Untuk dari program GUI-nya sendiri, masih terdapat beberapa bug. Bug-bug ini ada karena minimnya waktu untuk melakukan bug testing. Salah satu bug yang telah diketahui namun belum diperbaiki adalah error program yang terjadi ketika di suatu folder tidak ada folder atau file lainnya lagi.

V. Kesimpulan dan Saran

Dari tugas besar ini, kami belajar mengenai banyak hal, mulai bahasa pemrograman C# hingga softskill lainnya seperti pembagian waktu. Saya pribadi kurang baik di semua hal itu, termasuk kelalaian saya yang membuat saya tidak mendapat kelompok dan mengerjakan tugas ini sendirian. Dari saya pribadi, hal itu dapat diperbaiki.

Untuk tugasnya sendiri, menurut kami sudah menjadi tugas yang cukup baik untuk melatih keterampilan kami. Dengan tugas ini, kami menjadi lebih paham tentang bahasa pemrograman C# hingga tentang GUI.

Daftar Pustaka

- ❑ <http://michaeljulius11.blogspot.com/2017/10/pengertian-metode-pencarian-bfs-dan-dfs.html#:~:text=PENGERTIAN%20BFS%20>
- ❑ https://share.cocalc.com/share/5bc399842fed9875d75f21d8e9c505aad184b36d/LectureNotes/8_3-GraphTraversal.ipynb#:~:text=Graph%20Traversal%20merupakan%20proses%20menganjungi,titik%20asal%20ke%20titik%20tujuan.
- ❑ <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>
- ❑ <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>
- ❑ <https://www.geeksforgeeks.org/shortest-path-unweighted-graph/>