Laporan Tugas Besar 2
IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS dalam Implementasi *Folder Crawling*

Semester II Tahun 2021/2022

Disusun oleh:

Kelompok 5 - UTSduluBaruKerja

Vincent Ho	13520093
Ziyad Dhia Rafi	13520064
Jevant Jedidia Augustine	13520133

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

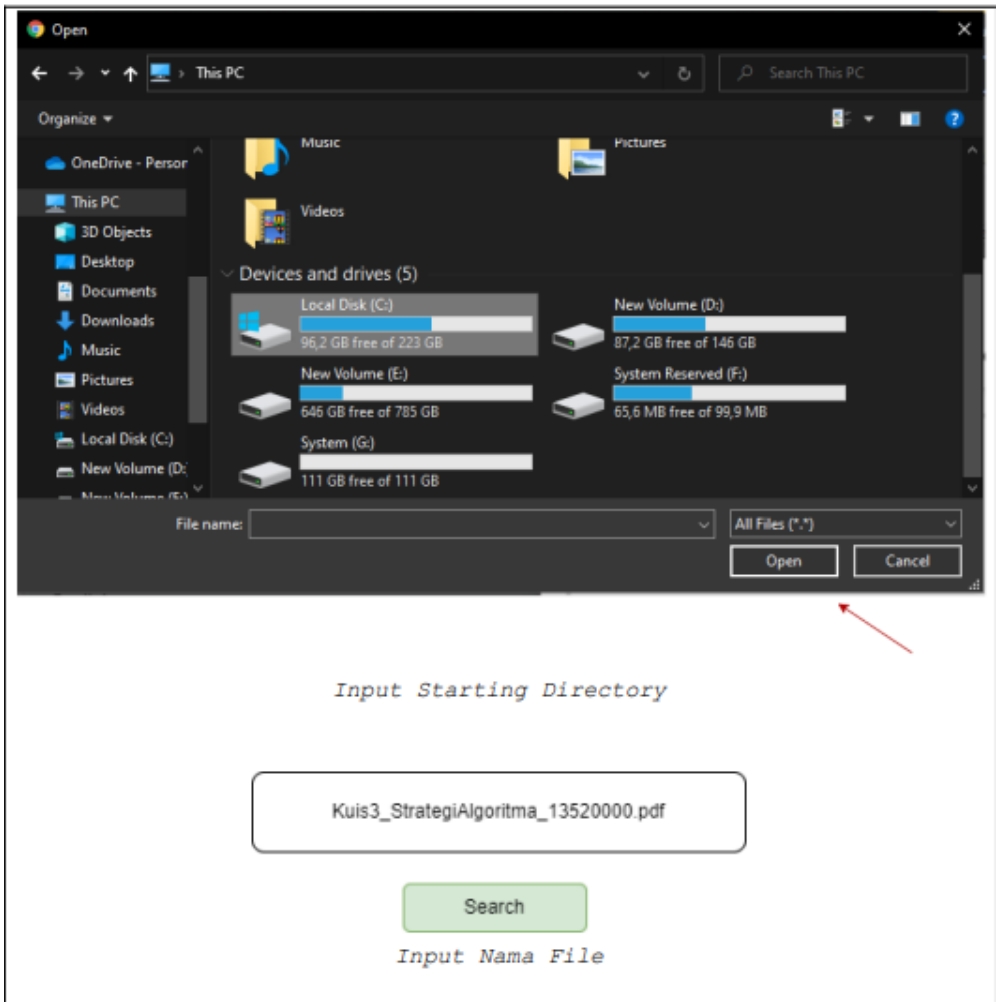
Bab I
Deskripsi Tugas

Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi GUI sederhana yang dapat memodelkan fitur dari file explorer pada sistem operasi, yang pada tugas ini disebut dengan Folder Crawling. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), Anda dapat menelusuri folder-folder yang ada pada direktori untuk mendapatkan direktori yang Anda inginkan. Anda juga diminta untuk memvisualisasikan hasil dari pencarian folder tersebut dalam bentuk pohon.

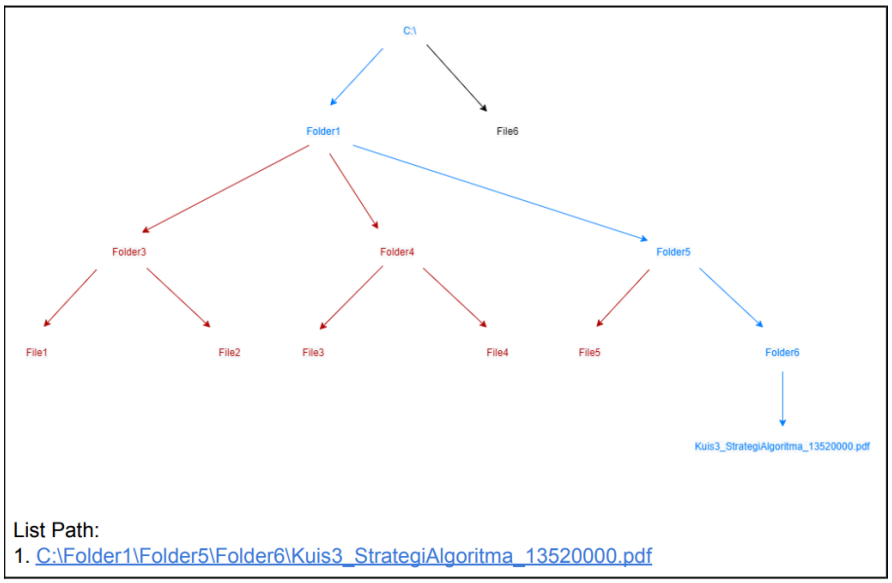
Selain pohon, Anda diminta juga menampilkan list path dari daun-daun yang bersesuaian dengan hasil pencarian. Path tersebut diharuskan memiliki hyperlink menuju folder parent dari file yang dicari, agar file langsung dapat diakses melalui browser atau file explorer. Contoh hal-hal yang dimaksud akan dijelaskan di bawah ini.

Contoh Input dan Output Program

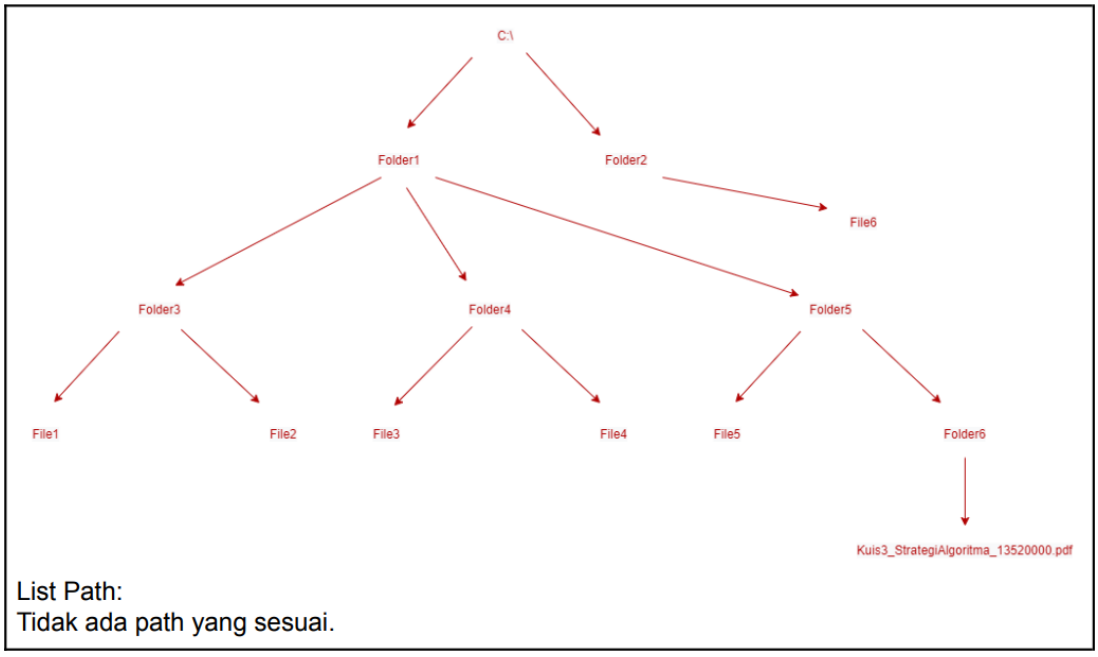
Contoh masukan aplikasi:



Contoh output aplikasi:



Misalnya pengguna ingin mengetahui langkah folder crawling untuk menemukan file Kuis3_StrategiAlgoritma_13520000.pdf.
Maka, path pencarian DFS adalah sebagai berikut. C:\ → Folder1 → Folder3 → File1 → Folder3 → File2 → Folder3 → Folder1 → Folder4 → File3 → Folder4 → File4 → Folder4 → Folder1 → Folder5 → File5 → Folder5 → Folder6 → Kuis3_StrategiAlgoritma_13520000.pdf.
Pada gambar di atas, rute yang dilewati pada pencarian DFS diwarnai dengan warna merah. Sedangkan, rute untuk menuju tempat file berada diberi warna biru. Rute yang masuk antrian tapi belum diperiksa diberi warna hitam. Anda bebas menentukan warnanya asalkan dibedakan antara ketiga hal tersebut.

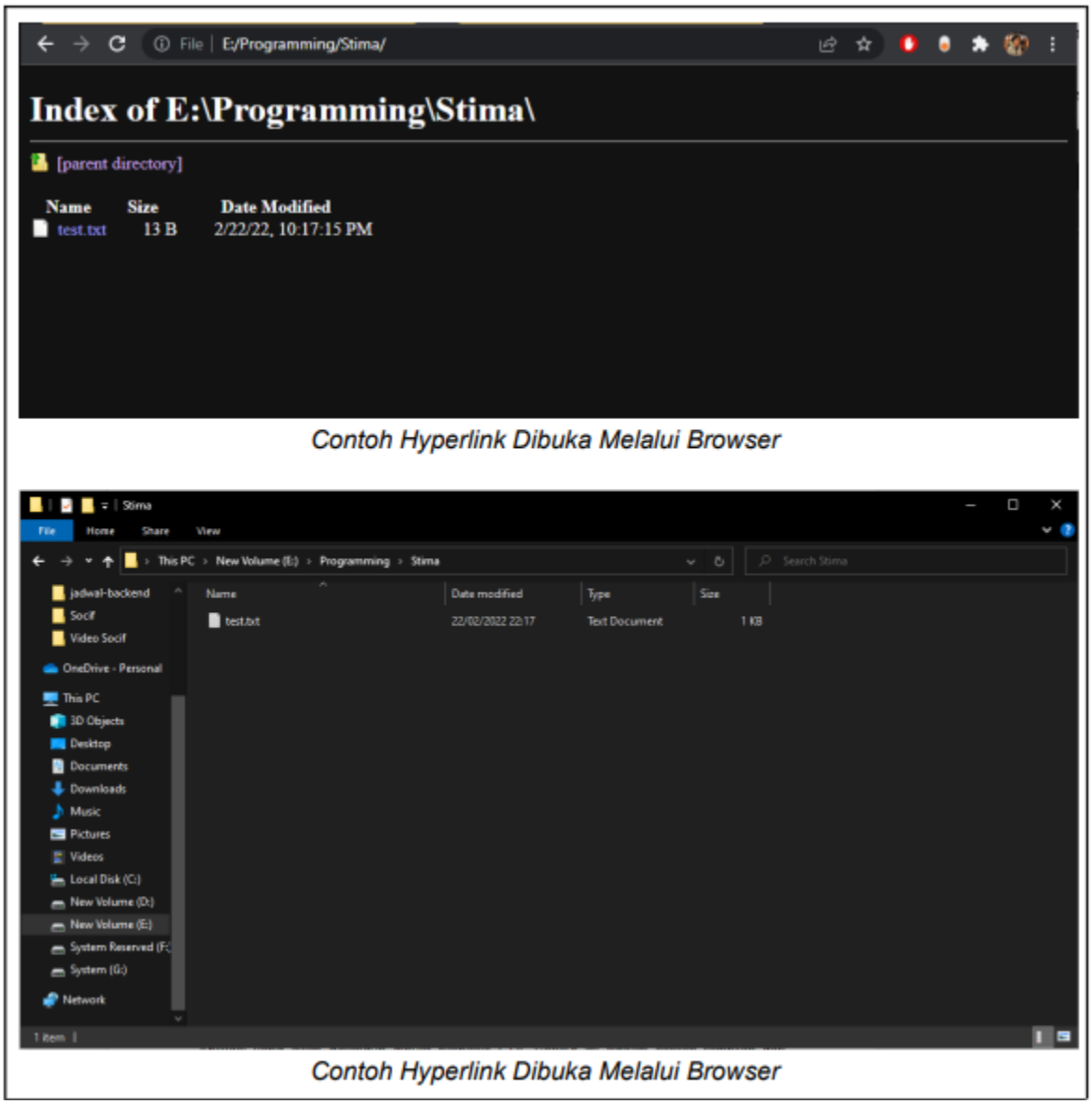


Jika file yang ingin dicari pengguna tidak ada pada direktori file, misalnya saat pengguna mencari Kuis3Probstat.pdf, maka path pencarian DFS adalah sebagai berikut: C:\ → Folder1 →

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



Contoh Hyperlink Dibuka Melalui Browser

Contoh Hyperlink Dibuka Melalui Browser

Bab II
Landasan Teori

Dasar Teori

Graph Traversal

Merupakan algoritma yang mengunjungi simpul-simpul pada graf dengan cara yang sistematik dengan asumsi bahwa graf yang dicari merupakan graf terhubung. Cara pencarian tersebut dibagi menjadi 2 macam, pencarian melebar dan pencarian mendalam. Graf merupakan representasi dari persoalan sedangkan traversal graf merupakan solusi dari persoalan tersebut.

Algoritma pencarian solusi dibagi menjadi dua:

- Tanpa informasi (*uninformed/blind search*) : dengan informasi tambahan (cth: BFS, DFS)
- Dengan informasi (*informed search*) : pencarian berbasis heuristik (Best First Search, A*)

Representasi graf dalam proses pencarian dibagi menjadi 2:

- Graf statis : graf yang sudah terbentuk sebelum proses pencarian dilakukan
- Graf dinamis : graf yang terbentuk saat proses pencarian dilakukan

BFS

Merupakan algoritma traversal graf tanpa informasi dimana pencarian dilakukan secara melebar.
Algoritma BFS:

1. Kunjungi simpul v
2. Kunjungi semua simpul yang merupakan tetangga dari simpul v
3. Kunjungi simpul yang belum dikunjungi yang bertetangga dengan simpul-simpul yang dikunjungi di langkah 2. Ulangi hingga semua simpul sudah dikunjungi.

DFS

Merupakan algoritma traversal graf tanpa informasi dimana pencarian dilakukan secara mendalam.
Algoritma DFS:

1. Kunjungi simpul v
2. Kunjungi simpul w yang merupakan tetangga dari simpul v
3. Ulangi DFS pada simpul w

4. Ketika mencapai simpul u sedemikian sehingga semua tetangga dari simpul u sudah dikunjungi, lakukan *backtrack* ke simpul yang memiliki tetangga simpul w yang belum dikunjungi
5. Pencarian DFS selesai apabila semua simpul yang ada sudah dikunjungi

C# Desktop Application Development

C# merupakan bahasa pemrograman yang berorientasi objek yang dikembangkan oleh perusahaan Microsoft sebagai bagian dari inisiatif framework .NET Framework pada tahun 2000. Bahasa C# memiliki beberapa fitur yang membantu dalam pembuatan aplikasi perangkat lunak. Beberapa fitur tersebut antara lain: automatic garbage collection yang dapat langsung melepas atau mengembalikan memori dari objek-objek yang tidak digunakan atau yang tidak dapat dicapai oleh program, exception handling yang dapat membantu untuk mendeteksi berbagai jenis error, memiliki library yang besar, execution speed yang cepat, dan masih banyak lagi.

Bab III
Analisis Pemecahan Masalah

Langkah-Langkah Pemecahan Masalah

Untuk mencari file yang diinginkan pada suatu folder *root* akan digunakan algoritma DFS dan BFS. Untuk DFS, apabila terdapat folder pada folder *root*, akan dilakukan pencarian DFS pada folder tersebut hingga pada folder yang dilakukan pencarian tidak terdapat lagi folder (hanya terdapat file atau kosong). File-file yang berada pada suatu folder akan ditelusuri apabila semua folder pada folder tersebut sudah dicari secara DFS. Setiap file akan diperiksa, apabila file yang diperiksa merupakan file yang dicari, maka file akan dimasukkan ke array solusi. Untuk pencarian yang menginginkan 1 kemunculan file yang diinginkan, pencarian DFS akan selesai dilakukan apabila file yang ingin dicari ditemukan atau tidak ada lagi folder yang belum dicari secara DFS. Untuk pencarian semua kemunculan file pencarian, DFS akan selesai apabila tidak ada lagi folder yang belum dicari secara DFS.

Pencarian BFS akan menggunakan queue (FIFO) untuk menyimpan folder-folder yang akan dicari atau ditelusuri. Untuk mencari sebuah file, maka algoritma BFS akan memasukkan semua folder pada level tersebut ke dalam queue. Kemudian akan diambil kepala dari queue tersebut yang merupakan sebuah folder dan diproses. Folder tersebut akan dijabarkan isinya (anaknya). Apabila anaknya merupakan folder, maka folder tersebut akan dimasukan ke queue, sedangkan apabila anaknya file, akan diperiksa apakah file tersebut merupakan file yang ingin dicari. Bila file tersebut merupakan file yang dicari, maka path dari file tersebut akan dimasukan ke array solusi. Pencarian BFS untuk 1 kemunculan akan selesai apabila file yang dicari telah ditemukan atau queue kosong, sedangkan untuk semua kemunculan akan selesai apabila queue kosong. Untuk kasus DFS dan BFS, isi pada array solusi merupakan path dari file yang dicari. Apabila array kosong, maka file yang dicari tidak ada pada folder root.

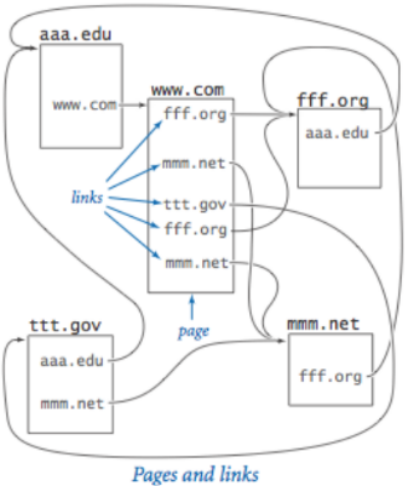
Pemetaan Persoalan

Algoritma penelusuran folder dari BFS akan menggunakan loop dan queue sedangkan untuk DFS akan menggunakan fungsi rekursif. Fungsi BFS dan DFS akan menerima masukan berupa path bertipe string dan mengembalikan sebuah pohon (graf traversal) yang merupakan pohon pencarian dari folder crawling (Root). Simpul pada pohon (nodes) merepresentasikan folder dan file, sedangkan sisi merepresentasikan bahwa sebuah folder X atau file Y berada di dalam sebuah folder Z apabila X dan Y merupakan anak dari Z. Mengetahui hal tersebut, maka hanya simpul folder yang dapat memiliki anak. Selama pencarian, array solusi yang terinisialisasi diluar fungsi BFS dan DFS dapat terisi dengan beberapa path dari file yang dicari atau kosong. Pencarian one occurrence akan dihentikan apabila telah ditemukan satu file goal, sedangkan pencarian all occurrence akan dihentikan apabila telah ditemukan semua file goal atau telah selesai mencari ke seluruh direktori dibawah root.

Ilustrasi Kasus Lain

Web Spider

Web Spider akan menjelajahi internet untuk mengunjungi halaman-halaman pada web secara periodik. Halaman pada web diilustrasikan sebagai graf berarah, dimana simpulnya merupakan halaman dari suatu website sedangkan sisinya merupakan link yang membuka suatu halaman website. Penelusuran akan dilakukan secara DFS. Pencarian DFS akan berhenti dilakukan apabila pada suatu halaman tidak ada lagi link yang mengarah ke website yang lain.



Ilustrasi Web Spider

Bab IV
Implementasi dan Pengujian

Implementasi Program

Pseudocode algoritma DFS

```
function DFSSearch (input string path) -> TreeNode
      DFS <- Tree //inisiasi sebuah tree
      DFS.root <- TreeNode(path, 1) //set root dari pohon dengan path
      folders = semua folder di dalam path
      files = semua files di dalam path

      if (tidak ada file dan tidak ada folder):
            //tambahkan node anak berisi pesan empty directory pada
            //pohon DFS
            DFS.root.AddChild("EMPTY DIRECTORY", 1)

      if (ada folder):
            foreach (folder in folders): //iterasi folder di folders
                  if (found = false):
                        //Hasil rekursif pada folder akan dijadikan
                        anak //dari pohon DFS
                        DFS.root.children.Add(DFSSearch(folder))
                  else:
                        //tambahkan node anak dengan isi folder pada
                        //pohon DFS
                        DFS.root.AddChild(folder,0);

      if (ada file):
            foreach (file in files): //iterasi file di files
                  if (found = false):
                        if (file merupakan file yang dicari):
                              //tambahkan file ke solusi
                              solution.add(file)
                              //tambahkan node anak dengan isi file pada
                              //pohon DFS
                              DFS.root.AddChild(file, 2)
                              if (pencarian hanya untuk 1 kemunculan):
                                    found <- true
                        else: //file bukan file yang dicari
                              //tambahkan node anak dengan isi file pada
                              //pohon DFS
                              DFS.root.AddChild(file, 1)
                  else://file sudah ditemukan
```

```
                    //tambahkan node anak dengan isi file pada
                    pohon //DFS
                    DFS.root.AddChild(file, 0)
    ->DFS.root
```

Pseudocode algoritma BFS

```
function BFSSearch (input string path) -> TreeNode
    queue.Add(path) //menambahkan path ke dalam queue
    nodes.Add(root) //menambahkan root ke dalam nodes

    while( panjang queue > 0 and not(found)):
        currentFolder = elemen paling depan dari queue
        queue.pop //elemen paling depan dari queue dibuang
        currentNode = elemen paling depan dari node
        node.pop //elemen paling depan dari node dibuang
        folders = semua folder yang ada di dalam currentFolder
        files = semua file yang ada di dalam currentFolder

        if (folders == NULL and files == NULL) then
            //tambahkan node berisi pesan empty directory pada
        solution tree
            currentNode.AddChild("EMPTY DIRECTORY", 1)

        if (folders != NULL)then
            //tambahkan setiap folder di dalam folders ke queue
            foreach(folder in folders)
                queue.Add(folder)
                folderName = nama dari folder
                newNode = membuat node baru dengan nama
            folderName
                nodes.Add(newNode) //menambahkan newNode
                kedalam Nodes
                //tambahkan node newNode pada solution tree
                currentNode.children.Add(newNode)

        if (files != NULL) then
            //periksa setiap file dalam files apakah ada yang
sama dengan goal file
            foreach(file in files)
                if (fileName == goal) then
                    //tambahkan node fileName pada solution
                    tree dan tandai sebagai solution
                    currentNode.AddChild(fileName, 2)
                    //solusi = path dari file tersebut
                    this.solution.Add(file)
```

```
                        if ( yang dicari bukan All Occurence) then
                              found = True
                              //jika yang dicari all occurence,
                  maka found tidak perlu dibikin menjadi true
                  saat ditemukan

                  else //fileName != goal
                        //tambahkan node fileName pada solution
                        tree
                        currentNode.AddChild(fileName,1)
return solution tree
```

Struktur Data

Untuk program Folder Crawling, akan digunakan struktur data berbasis objek.

1. Program.cs
   Kelas yang akan menjalankan program keseluruhan

2. Form1.cs
   Kelas yang mengatur dan menampilkan GUI dari program

3. Path.cs
   Kelas yang berisikan fungsi-fungsi yang akan membantu dalam pencarian file baik pada BFS maupun DFS.
   - FoldersInPath
     Menerima masukan berupa path dan mengembalikan folders yang berada di path tersebut.
   - FilesInPath
     Menerima masukan berupa path dan mengembalikan files yang berada di path tersebut.
   - removePath
     Menerima masukan berupa path dari suatu folder atau file dan mengembalikan nama file atau folder tanpa path lengkapnya.
   - addSolution
     Menerima masukan berupa list of solution, folder root, dan akar dari pohon (BFS maupun DFS). Prosedur akan mengubah warna dari jalur yang merupakan solusi pada pohon BFS dan DFS menjadi warna biru dengan bantuan prosedur goalPath.
   - goalPath
     Menerima masukan berupa array of folder dan TreeNode. Prosedur akan mencari anak dari TreeNode yang memiliki nama yang sama dengan elemen

pertama dari array of folder dan mengubah node tersebut menjadi warna biru kemudian memanggil prosedur goalPath dengan masukan array of folder yang elemen pertamanya sudah dihapus dan node yang warnanya diubah.

4. Tree.cs
   Kelas yang membangun pohon pencarian beserta nodenya.
   a. Kelas TreeNode
      Atribut:
      - String name
        Nama dari node.
      - Int id
        Id dari node.
      - Int category
        Kategory dari node, 0 artinya node berada di dalam queue, 1 artinya node sudah diperiksa, 2 artinya node merupakan jalur menuju solusi.
      - List of TreeNode children
        Anak-anak dari node.

      Method:
      - TreeNode(string name, int category)
        Konstruktor dari kelas TreeNode dengan nama dan kategori yang terdefinisi.
      - AddChild(string name, int category)
        Membuat anak node baru dengan nama dan kategori yang terdefinisi dan menambahkannya sebagai anak dari node root.
      - SetCategory(int category)
        Mengubah kategori dari node.
      - SetName(string name)
        Mengubah nama dari node.

   b. Kelas Tree
      Atribut:
      - TreeNode root
        Akar dari pohon yang merupakan node.

      Method:
      - Tree
        Konstruktor dari kelas Tree. Menginisiasi root dengan nilai null.
      - Print (TreeNode t)
        Mencetak akar dari pohon kemudian memanggil PrintLevel.
      - PrintLevel(TreeNode t, int level)
        Mencetak anak dari akar pohon.

5. DFS.cs
   Kelas yang akan melakukan folder crawling dengan algoritma DFS.
   Atribut:
   - String root

Path folder awal yang akan digunakan untuk folder crawling.
- String goal
  Nama file yang akan dicari.
- List of string solution
  Path yang merupakan solusi dari pencarian DFS.
- Tree DFSTree
  Pohon yang dihasilkan dari pencarian DFS.
- Boolean found
  Menandakan apakah file sudah ditemukan atau belum.
- Boolean allOcc
  Menandakan apakah pencarian merupakan 1 kemunculan atau semua kemunculan

Method:
- DFS(string root, string goal, boolean allOcc)
  Konstruktor kelas DFS dengan root, goal, dan allOcc terdefinisi.
- DFSRecursive(string path)
  Fungsi rekursif yang akan melakukan pencarian file dengan algoritma DFS dengan masukan suatu path. Akan mengembalikan TreeNode yang telah ditelusuri.

6. BFS.cs
   Kelas yang akan melakukan folder crawling dengan algoritma BFS.
   Atribut:
   - List of string queue
     Queue yang akan menampung folder-folder yang dicari berdasarkan algoritma BFS.
   - String root
     Path folder awal yang akan digunakan untuk folder crawling.
   - String goal
     Nama file yang akan dicari.
   - List of string solution
     Path yang merupakan solusi dari pencarian DFS.
   - List of TreeNode node
     Queue yang menampung TreeNode yang dimasukkan secara FILO.
   - Tree DFSTree
     Pohon yang dihasilkan dari pencarian DFS.
   - Boolean found
     Menandakan apakah file sudah ditemukan atau belum.
   - Boolean allOcc
     Menandakan apakah pencarian merupakan 1 kemunculan atau semua kemunculan

   Method:
   - BFS(string root, string goal, boolean allOcc)
     Konstruktor kelas DFS dengan root, goal, dan allOcc terdefinisi.

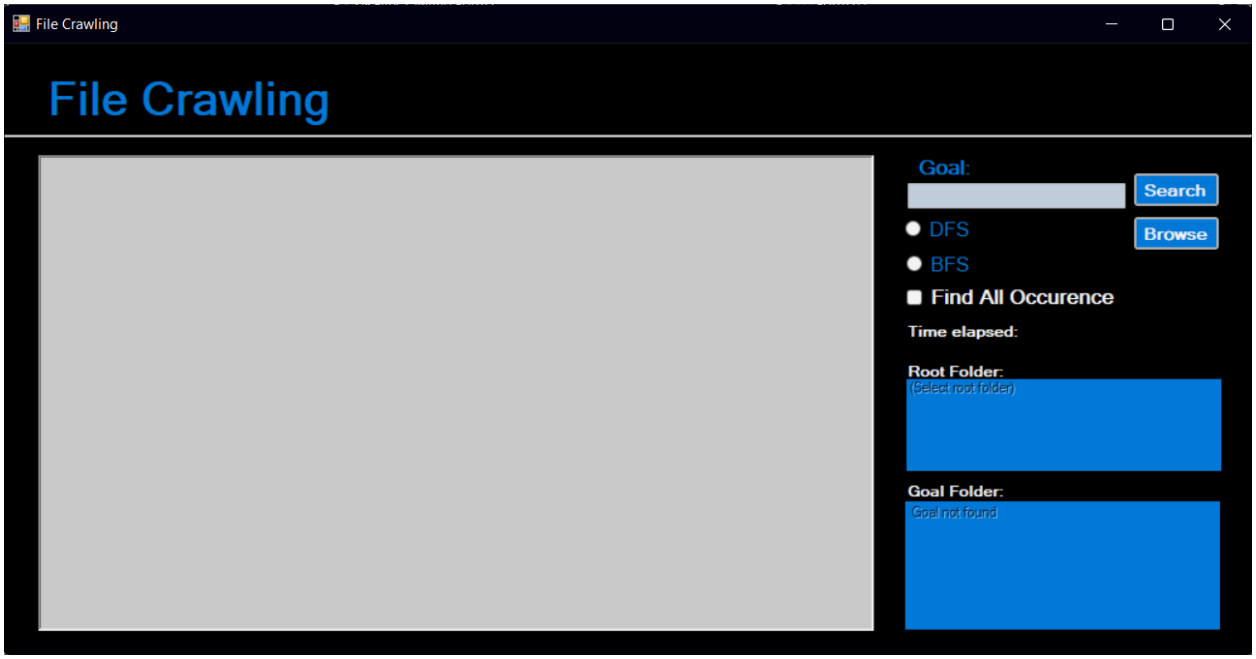
- BFSSearch(string path)
  Fungsi yang menerima masukan path, melakukan pencarian BFS terhadap path untuk menemukan file yang ingin dicari, dan mengembalikan sebuah TreeNode yang telah dicari secara BFS.

Tata Cara Penggunaan Program

Run

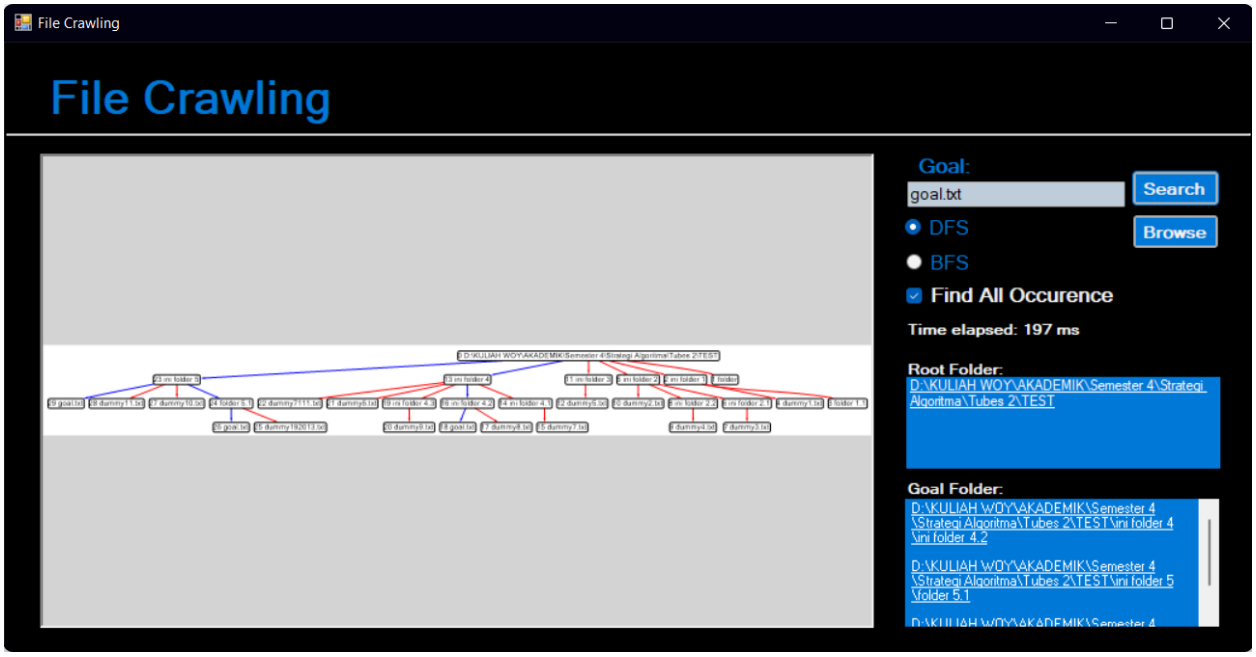
1. Pada folder exe, buka setup.exe untuk melakukan instalasi
2. Setelah selesai, jalankan FileCrawling.application untuk memulai program

Menggunakan Aplikasi



Tampilan utama aplikasi

1. Tekan browse untuk membuka directory akar untuk pencarian, setelah dipilih, folder akar akan muncul di bagian Root folder, dan bisa diklik untuk membuka di file explorer
2. Masukkan goal text untuk mencari file dengan nama yang sama persis dengan goal (termasuk ekstensi)
3. Pilih opsi yang digunakan untuk pencarian, Depth First Search (DFS) atau Breadth First Search (BFS)
4. Centang Find All Occurrence untuk mencari semua file yang sesuai, hapus centang bila hanya ingin mencari satu
5. Tekan tombol search untuk memulai pencarian

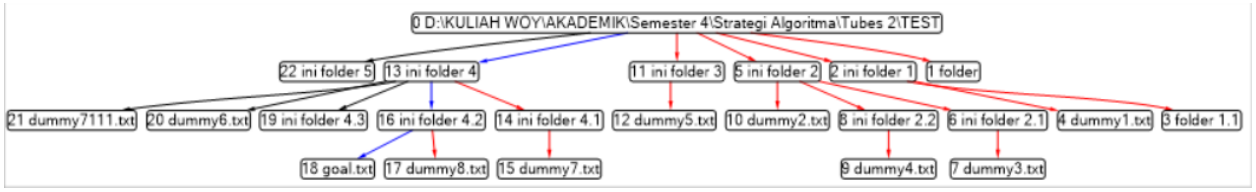


Tampilan setelah pencarian selesai

6. Setelah pencarian selesai, akan ditampilkan waktu yang digunakan untuk menjalankan pencarian
7. Apabila file ditemukan, tersedia link untuk membuka folder dari file tersebut pada bagian Goal Folder

Hasil Pengujian

DFS - Test Case 1: “goal.txt”

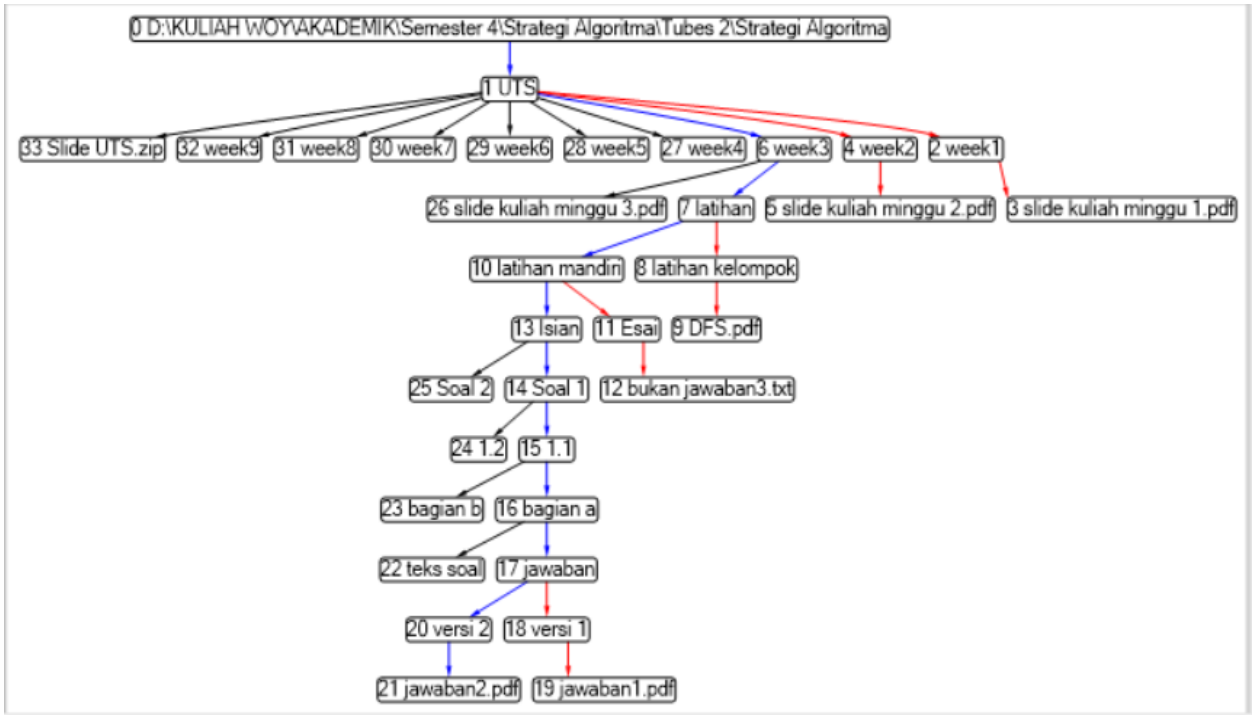


DFS- Test Case 1 All Occurence: “goal.txt”

BFS - Test Case 1: “goal.txt”

BFS - Test Case 1 All Occurence: “goal.txt”

DFS - Test Case 2: “jawaban2.pdf”



DFS- Test Case 2 All Occurence: “jawaban2.pdf”

BFS - Test Case 2: “jawaban2.pdf”

BFS - Test Case 2 All Occurence: “jawaban2.pdf”

Analisis dari Desain Solusi

Berdasarkan hasil pengujian yang dilakukan dapat dilihat bahwa BFS dan DFS terlihat memiliki efisiensi yang serupa. Perbedaan antara kedua algoritma pencarian tersebut adalah BFS akan mencari secara menyamping terlebih dahulu atau mengutamakan nodes yang memiliki level yang sama terlebih dahulu, sehingga apabila file yang dicari terletak pada direktori yang tidak dalam maka pencarian menggunakan algoritma BFS akan mendapatkan hasil yang lebih cepat. Sedangkan algoritma pencarian DFS akan mencari ke dalam atau ujung sebuah direktori terlebih dahulu, sehingga apabila file yang dicari terletak pada direktori yang dalam maka pencarian menggunakan algoritma DFS akan mendapatkan hasil yang lebih cepat.

Kedua strategi memiliki keunggulannya masing-masing, contohnya dalam menemukan “jawaban2.pdf” yang memiliki posisi lebih dalam (terdapat 2 file jawaban2.pdf) pada test case 2, algoritma DFS membutuhkan langkah lebih sedikit untuk menemukannya karena mengutamakan penelusuran ke dalam terlebih dahulu baru ke samping, sedangkan untuk menemukan file “jawaban2.pdf” yang sama menggunakan algoritma BFS akan memakan langkah jauh lebih banyak. Sebaliknya untuk menemukan file “jawaban2.pdf” yang memiliki posisi lebih luar pada test case 2, algoritma BFS membutuhkan langkah lebih sedikit untuk menemukannya karena menelusuri node yang selevel terlebih dahulu, sedangkan untuk menemukan file “jawaban2.pdf” yang sama menggunakan algoritma DFS akan memakan langkah yang lebih banyak.

Bab V
Kesimpulan dan Saran

Kesimpulan

Kesimpulan yang dapat diambil dari implementasi algoritma BFS dan DFS pada implementasi folder crawling adalah algoritma BFS dan DFS dapat diimplementasikan dengan baik jika sudah mengerti pemetaan permasalahan dan alur serta strategi pencarian dari algoritma tersebut. Implementasi folder crawling dengan kedua strategi tersebut dapat menghasilkan jalur solusi dan kecepatan pencarian yang berbeda pula. Pemilihan dapat bergantung pada seberapa dalam letak file yang ingin dicari. Jika file yang ingin dicari terletak pada posisi yang tidak terlalu dalam maka algoritma BFS memiliki kemungkinan untuk menemukan file goal tersebut lebih cepat, namun jika file yang ingin dicari terletak pada posisi yang sangat dalam maka algoritma DFS memiliki kemungkinan untuk menemukan file goal tersebut lebih cepat.

Saran

Berdasarkan hasil kerja kelompok kami, terdapat beberapa saran yang dapat digunakan untuk membantu pengimplementasian algoritma BFS dan DFS dalam folder crawling:

1. Diperlukan pemahaman yang baik mengenai konsep algoritma BFS dan DFS sehingga memudahkan dalam mengimplementasi program folder crawling.
2. Sering membaca dokumentasi dari C# karena merupakan bahasa pemrograman baru bagi sebagian besar orang sehingga dapat membantu dalam proses pengerjaan.

Video demo program dapat diakses pada link berikut :
https://www.youtube.com/watch?v=jHKLVcAfwF0

Source code program dapat diakses pada link berikut :
https://github.com/vincen-tho/Directory-Crawler

Daftar Pustaka

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