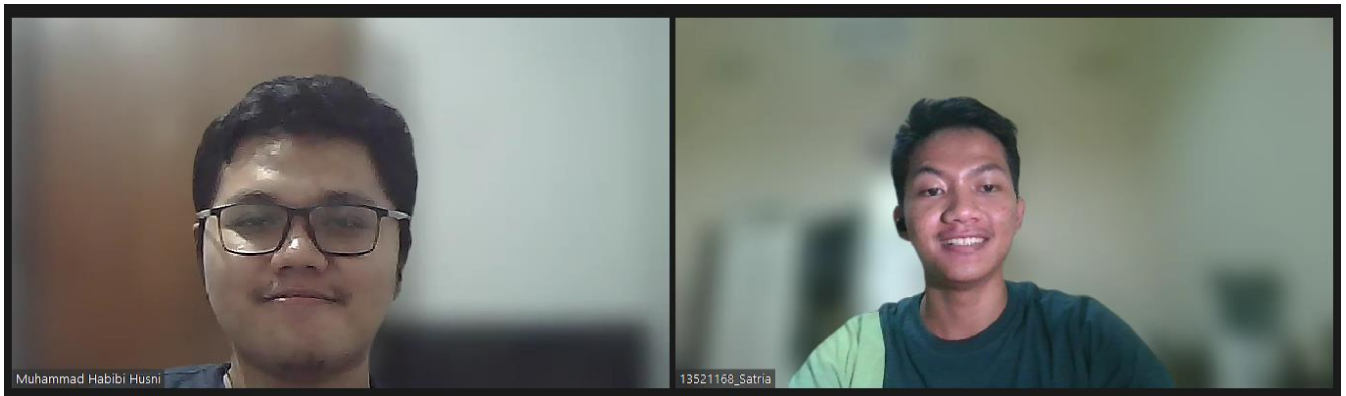


LAPORAN TUGAS BESAR II

IF2211 Strategi Algoritma

Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan *Maze Treasure Hunt*



Dipersiapkan oleh:

Kelompok 23 -
MazeTreasureHunt

Muhammad Habibi Husni	13521169
Satria Octavianus Nababan	13521168
Muhammad Dhiwaul Akbar	13521158

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

Daftar Isi	2
BAB I.....	3
Deskripsi Tugas	3
BAB II.....	7
Landasan Teori	7
A. Graph Traversal	7
B. BFS	7
C. DFS	8
D. C# Desktop Application Development	9
BAB III	11
Analisis Pemecahan Masalah.....	11
A. Langkah-Langkah Pemecahan Masalah.....	11
B. <i>Mapping</i> Persoalan menjadi Elemen Algoritma BFS dan DFS	11
C. Ilustrasi Kasus Lain.....	12
BAB IV	14
Implementasi dan Pengujian	14
A. Link Repository Github	14
B. Link Video Youtube	14
C. Implementasi Program (Pseudocode)	14
D. Struktur Data dan Spesifikasi Program.....	16
E. Tata Cara Penggunaan Program.....	17
F. Fitur-Fitur Yang Tersedia Pada Aplikasi	19
G. Hasil Pengujian	20
H. Analisis Desain Solusi	27
BAB V	28
Penutup	28
A. Kesimpulan	28
B. Saran	28
C. Refleksi	28
D. Tanggapan Terkait Tugas Besar Ini	28
Daftar Pustaka.....	29

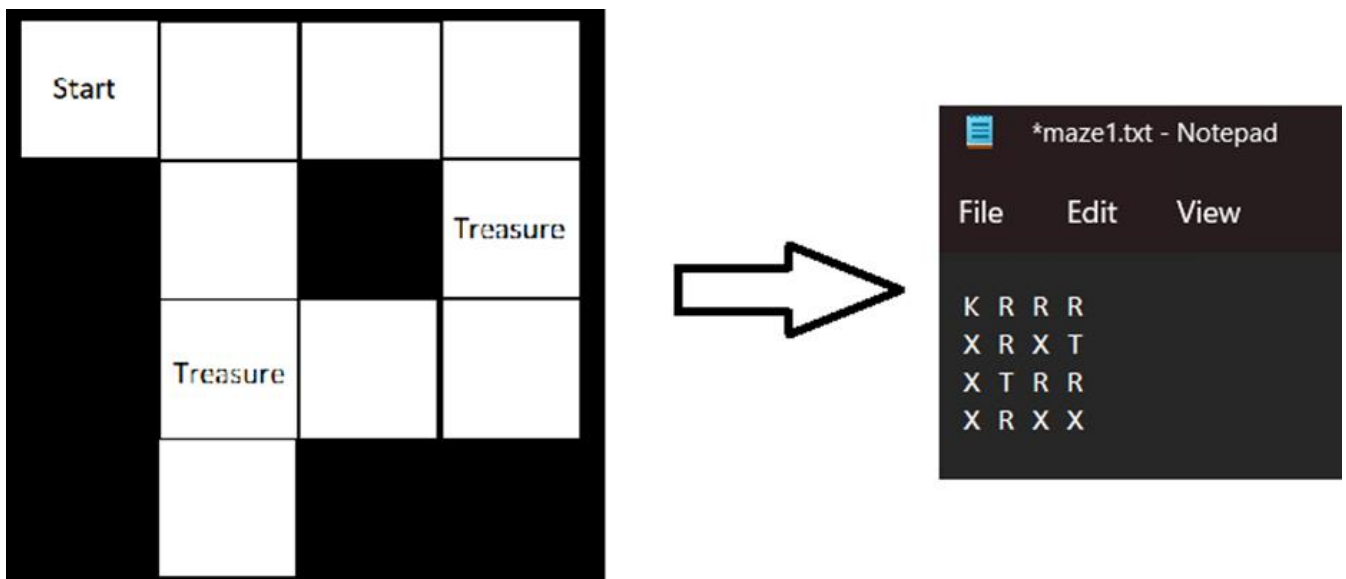
BAB I

Deskripsi Tugas

Dalam tugas besar ini, diminta untuk membangun sebuah aplikasi dengan GUI sederhana yang dapat mengimplementasikan BFS dan DFS untuk mendapatkan rute memperoleh seluruh *treasure* atau harta karun yang ada. Program dapat menerima dan membaca *input* sebuah file txt yang berisi *maze* yang akan ditemukan solusi rute mendapatkan *treasure*-nya. Untuk mempermudah, batasan dari *input maze* cukup berbentuk segi-empat dengan spesifikasi simbol sebagai berikut :

- K : Krusty Krab (Titik awal)
- T : *Treasure*
- R : *Grid* yang mungkin diakses / sebuah lintasan
- X : *Grid* halangan yang tidak dapat diakses

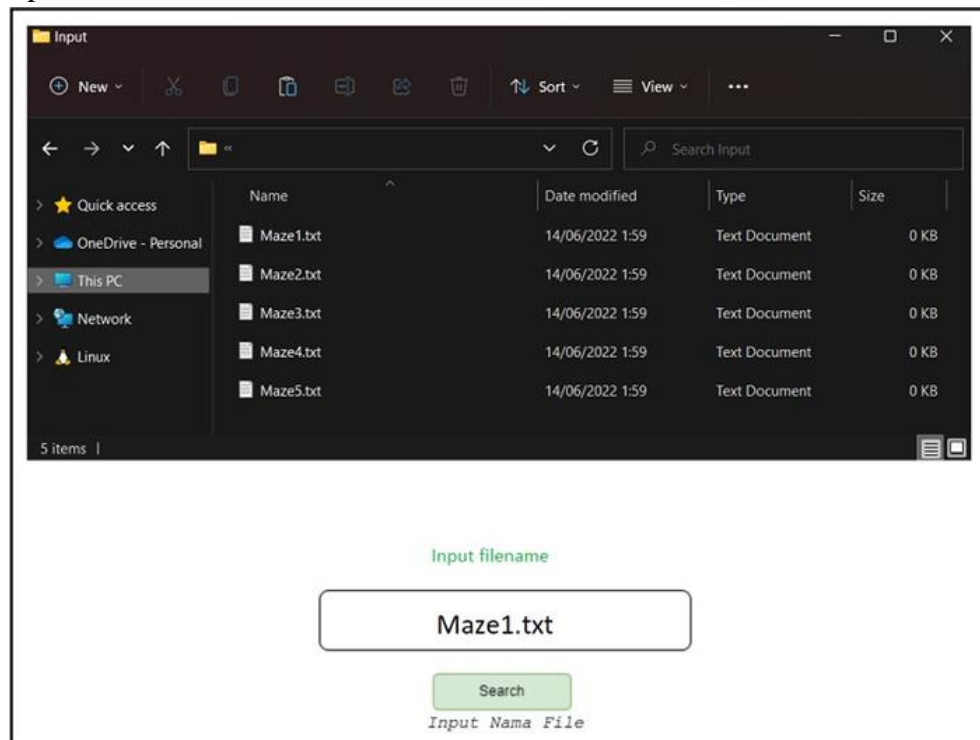
Contoh file *input* :



Gambar 1.1 Ilustrasi input file maze

Dengan memanfaatkan algoritma *Breadth First Search* (BFS) dan *Depth First Search* (DFS), anda dapat menelusuri *grid* (simpul) yang mungkin dikunjungi hingga ditemukan rute solusi, baik secara melebar ataupun mendalam bergantung alternatif algoritma yang dipilih. **Rute solusi adalah rute yang memperoleh seluruh *treasure* pada *maze*.** Perhatikan bahwa rute yang diperoleh dengan algoritma BFS dan DFS dapat berbeda, dan banyak langkah yang dibutuhkan pun menjadi berbeda. Prioritas arah simpul yang dibangkitkan dibebaskan asalkan ditulis di laporan ataupun readme, semisal LRUD (left right up down). **Tidak ada pergerakan secara diagonal.** Anda juga diminta untuk memvisualisasikan *input* txt tersebut menjadi suatu *grid maze* serta hasil pencarian rute solusinya. Cara visualisasi *grid* dibebaskan, sebagai contoh dalam bentuk matriks yang ditampilkan dalam GUI dengan keterangan berupa teks atau warna. Pemilihan warna dan maknanya dibebaskan ke masing - masing kelompok, asalkan dijelaskan di readme / laporan.

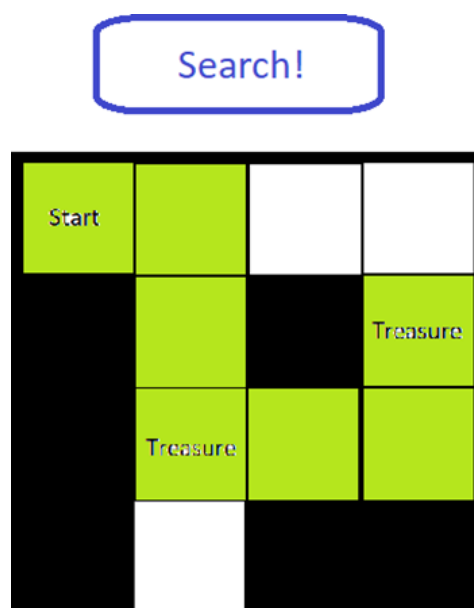
Contoh *input* aplikasi :



Gambar 1.2 Contoh input program

Daftar *input maze* akan dikemas dalam sebuah folder yang dinamakan test dan terkandung dalam repository program. Folder tersebut akan setara kedudukannya dengan folder src dan doc (struktur folder repository akan dijelaskan lebih lanjut di bagian bawah spesifikasi tubes). Cara *input maze* boleh langsung *input* file atau dengan *textfield* sehingga pengguna dapat mengetik nama *maze* yang diinginkan. Apabila dengan *textfield*, harus handle kasus apabila tidak ditemukan dengan nama file tersebut.

Contoh output Aplikasi :



Gambar 1.3 Contoh output program untuk gambar 2

Setelah program melakukan pembacaan *input*, program akan memvisualisasikan *grid*nya terlebih dahulu tanpa pemberian rute solusi. Hal tersebut dilakukan agar pengguna dapat mengerjakan terlebih dahulu *treasure hunt* secara manual jika diinginkan. Kemudian, program menyediakan tombol *solve* untuk mengeksekusi algoritma DFS dan BFS. Setelah tombol diklik, program akan melakukan pemberian warna pada rute solusi.

Spesifikasi Program:

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

Gambar 1.4 Tampilan Program Sebelum dicari solusinya

Gambar 6. Tampilan Program setelah dicari solusinya

Catatan: Tampilan diatas hanya berupa contoh *layout* dari aplikasi saja, untuk design *layout* aplikasi dibebaskan dengan syarat mengandung seluruh *input* dan output yang terdapat pada spesifikasi.

Spesifikasi GUI:

1. **Masukan program** adalah file *maze treasure hunt* tersebut atau nama filenya.
2. Program dapat menampilkan visualisasi dari *input* file *maze* dalam bentuk *grid* dan pewarnaan

sesuai deskripsi tugas.

3. Program memiliki *toggle* untuk menggunakan alternatif algoritma BFS ataupun DFS.
4. Program memiliki tombol *search* yang dapat mengeksekusi pencarian rute dengan algoritma yang bersesuaian, kemudian memberikan warna kepada rute solusi output.
5. **Luaran program** adalah banyaknya *node (grid)* yang diperiksa, banyaknya langkah, rute solusinya, dan waktu eksekusi algoritma.
6. **(Bonus)** Program dapat menampilkan progress pencarian *grid* dengan algoritma yang bersesuaian. Hal tersebut dilakukan dengan memberikan *slider / input box* untuk menerima durasi jeda tiap *step*, kemudian memberikan warna kuning untuk tiap *grid* yang sudah diperiksa dan biru untuk *grid* yang sedang diperiksa.
7. **(Bonus)** Program membuat *toggle* tambahan untuk persoalan TSP. Jadi apabila *toggle* dinyalakan, rute solusi yang diperoleh juga harus kembali ke titik awal setelah menemukan segala harta karunnya (Tetap dengan algoritma BFS atau DFS).
8. GUI dapat dibuat **sekreatif** mungkin asalkan memuat 5 (7 jika mengerjakan bonus) spesifikasi di atas.

BAB II

Landasan Teori

A. Graph Traversal

Graf adalah sekumpulan objek terstruktur dimana beberapa pasangan objek mempunyai hubungan atau keterkaitan tertentu. Graf terdiri dari himpunan objek-objek yang dinamakan titik, simpul, atau sudut, yang dihubungkan oleh penghubung yang disebut garis atau sisi. Traversal adalah proses kunjungan dalam graf atau pohon, dengan setiap node hanya dikunjungi tepat satu kali.

Traversal graf memiliki arti mengunjungi simpul-simpul dengan cara yang sistematis. Dalam traversal graf ini terdapat dua cara, yaitu:

1. Pencarian Melebar (*Breadth First Search / BFS*)
2. Pencarian Mendalam (*Depth First Search / DFS*)

Dalam kedua cara traversal graf ini, digunakan asumsi bahwa tiap simpul/node/titik setidaknya terhubung dengan satu simpul/node/titik lainnya dan graf merupakan graf terhubung. Traversal graf ini termasuk algoritma pencarian solusi, dimana algoritma pencarian solusi ini terbagi menjadi dua jenis, yaitu:

1. Tanpa Informasi (*uninformed / blind search*)

Algoritma pencarian tanpa informasi ini dilakukan tanpa ada informasi tambahan. Contohnya adalah DFS, BFS, *Depth Limited Search*, *Iterative Deepening Search*, *Uniform Cost Search*.

2. Dengan Informasi (*informed search*)

Algoritma pencarian dengan informasi ini merupakan pencarian berbasis heuristik. Pada algoritma ini, diketahui *non-goal state* yang “lebih menjanjikan” daripada yang lain. Contohnya adalah *Best First Search* dan *A**.

Dalam proses pencarian atau pengunjungan semua node pada graf, terdapat dua pendekatan:

1. Graf Statis

Pada pendekatan graf statis ini, graf sudah diciptakan sebelum proses pencarian dilakukan.

2. Graf Dinamis

Pada pendekatan graf dinamis ini, graf baru dibuat saat proses pencarian sedang dilakukan. Sebelum pencarian, tidak tersedia graf.

B. BFS

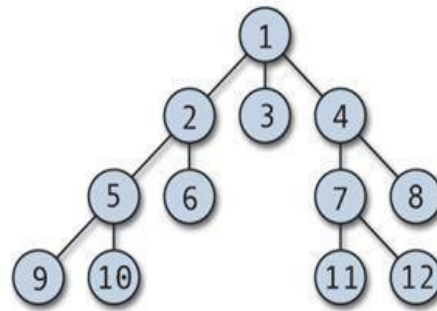
Algoritma BFS (*Breadth First Search*) adalah salah satu algoritma yang digunakan untuk pencarian jalur. Algoritma ini adalah salah satu algoritma pencarian jalur sederhana, dimana pencarian dimulai dari simpul awal, kemudian dilanjutkan ke semua simpul yang bertetangga dengan simpul awal secara terurut. Jika simpul tujuan belum ditemukan, maka perhitungan akan diulang lagi ke masing-masing simpul cabang dari masing-masing simpul, sampai simpul tujuan ditemukan.

Breadth First Search (BFS) adalah metode traversing yang digunakan dalam grafik.

Ini menggunakan antrian untuk menyimpan simpul yang dikunjungi. Dalam metode ini

yang ditekankan adalah pada simpul grafik, satu simpul dipilih pada awalnya kemudian dikunjungi dan ditandai. Simpul yang berdekatan dengan simpul yang dikunjungi kemudian dikunjungi dan disimpan dalam antrian berurutan. Demikian pula, simpul yang disimpan kemudian diperlakukan satu per satu, dan simpul yang berdekatan dikunjungi. Sebuah node sepenuhnya dieksplorasi sebelum mengunjungi node lain dalam grafik, dengan kata lain, node tersebut melintasi node yang belum dieksplorasi paling dangkal terlebih dahulu.

Berikut ini adalah urutan pengunjungan suatu node pada *graph* (pohon) menggunakan algoritma BFS:



Gambar 6 Contoh BFS pada pohon

(sumber: <https://onbuble.wordpress.com/2011/05/26/6/>)

Dalam algoritma BFS, simpul anak yang telah dikunjungi disimpan dalam suatu antrian. Antrian ini digunakan untuk mengacu simpul-simpul yang bertetangga dengannya yang akan dikunjungi kemudian sesuai urutan pengantrian. Berikut ini langkah-langkah algoritma BFS:

1. Masukkan simpul ujung/akar/root ke dalam antrian.
2. Ambil simpul dari awal antrian, lalu cek apakah simpul merupakan solusi.
3. Jika simpul merupakan solusi, pencarian selesai dan hasil dikembalikan.

Jika simpul bukan solusi, seluruh simpul yang bertetangga dengan simpul tersebut. (simpul anak) masuk ke dalam antrian secara berurutan.

1. Jika antrian kosong dan setiap simpul sudah dicek, pencarian selesai dan mengembalikan hasil solusi tidak ditemukan.
2. Ulangi pencarian dari langkah kedua.

C. DFS

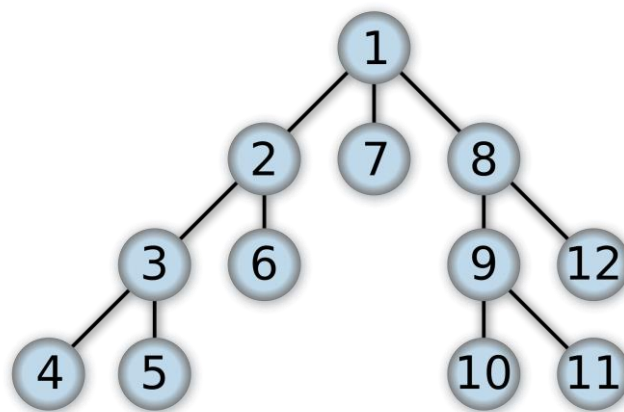
DFS (Depth First Search) adalah salah satu algoritma penelusuran struktur graf / pohon berdasarkan kedalaman. Simpul ditelusuri dari root kemudian ke salah satu simpul anaknya (misalnya prioritas penelusuran berdasarkan anak pertama [simpul sebelah kiri]), maka penelusuran dilakukan terus melalui simpul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

Dalam implementasinya DFS dapat diselesaikan dengan cara rekursif atau dengan bantuan struktur data stack. Langkah-langkah DFS dengan metode rekursif adalah sebagai berikut:

1. Kunjungi simpul v (simpul awal / root)
2. Kunjungi simpul w yang bertetangga dengan simpul v
3. Ulangi DFS langkah 1 dengan w sebagai simpul awal
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 bila tidak ada lagi simpul yang belum dikunjungi yang dapat dari simpul yang telah dikunjungi.

Berikut ini adalah urutan pengunjungan suatu node pada *graph* (pohon) menggunakan algoritma DFS:



Gambar 7 Contoh DFS pada pohon

(sumber: https://en.wikipedia.org/wiki/Depth-first_search#/media/File:Depth-first-tree.svg)

D. C# Desktop Application Development

C# *Desktop Application Development* mengacu pada proses pembuatan aplikasi desktop menggunakan bahasa pemrograman C# dan kerangka kerja .NET. C# adalah bahasa pemrograman berorientasi objek modern yang dikembangkan oleh Microsoft. Ini banyak digunakan untuk mengembangkan aplikasi desktop, aplikasi web, dan game. Kerangka kerja .NET menyediakan kumpulan pustaka dan alat yang kaya untuk membangun berbagai jenis aplikasi.

Beberapa alat dan teknologi populer untuk pengembangan aplikasi desktop C# termasuk Microsoft Visual Studio, Windows Forms, Windows Presentation Foundation (WPF), dan Universal Windows Platform (UWP). Alat-alat ini menyediakan berbagai fitur dan kemampuan untuk membuat aplikasi desktop yang kaya dan menarik. Secara keseluruhan, pengembangan aplikasi desktop C# menawarkan platform yang kuat dan fleksibel untuk membangun aplikasi desktop yang dapat berjalan di komputer berbasis Windows.

Pada Tugas Besar kali ini digunakan Window Form Application untuk merealisasikan GUI (*Graphical User Interface*) dari program yang dibuat. Dengan memanfaatkan Visual Studio,

langkah-langkah untuk membuatnya adalah sebagai berikut:

1. Unduh Visual Studio
2. *Install* dan jalankan Visual Studio
3. Pada halaman utama, pilih “Create a new project”
4. Pilih Windows Forms App sebagai template dari aplikasi yang ingin dibuat
5. Tuliskan nama dari aplikasi dan pilih lokasi penyimpanan dari aplikasi yang Anda buat pada window yang muncul
6. Pilih jenis *Framework* yang ingin digunakan
7. Ketik “Create”
8. Lalu akan muncul *default* form dari Windows Form.
9. Untuk menjalankan aplikasi, klik tombol panah ke kanan berwarna hijau. Visual studio akan melakukan *build* terhadap aplikasi dan jika berhasil, .exe akan secara otomatis dijalankan.

BAB III

Analisis Pemecahan Masalah

A. Langkah-Langkah Pemecahan Masalah

Dari permasalahan yang disebutkan pada Bab 1, telah dilakukan beberapa langkah

- langkah pemecahan masalah antara lain:

- Memahami permasalahan yang diberikan pada Bab 1
- Memahami konsep algoritma BFS dan DFS dalam mencari *treasure* sesuai dengan batasan yang diberikan
- Membagi permasalahan menjadi elemen BFS dan DFS
- Mempelajari bahasa pemrograman C# dan melakukan instalasi Visual Studio sebagai IDE yang akan digunakan untuk melakukan pemrograman dengan bahasa C#
- Melakukan testing terhadap algoritma BFS dan DFS yang telah didesain
- Membuat GUI dalam Windows Forms Application yang dapat menerima masukkan pengguna berupa file *case*, pilihan algoritma (BFS atau DFS), dan *toggle* untuk menggunakan TSP serta mengatur durasi jeda setiap *step* pencarian *treasure* pada *progres bar*.
- Membuat visualisasi rute *grid* pencarian *treasure* pada GUI berdasarkan masukkan file *case* dan pilihan algoritma.
- Melakukan testing terhadap GUI yang telah dibuat


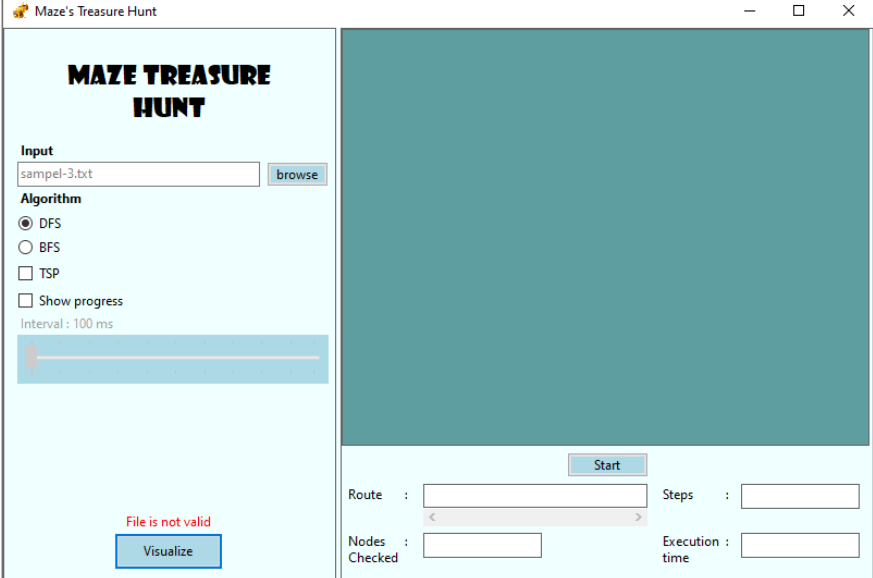

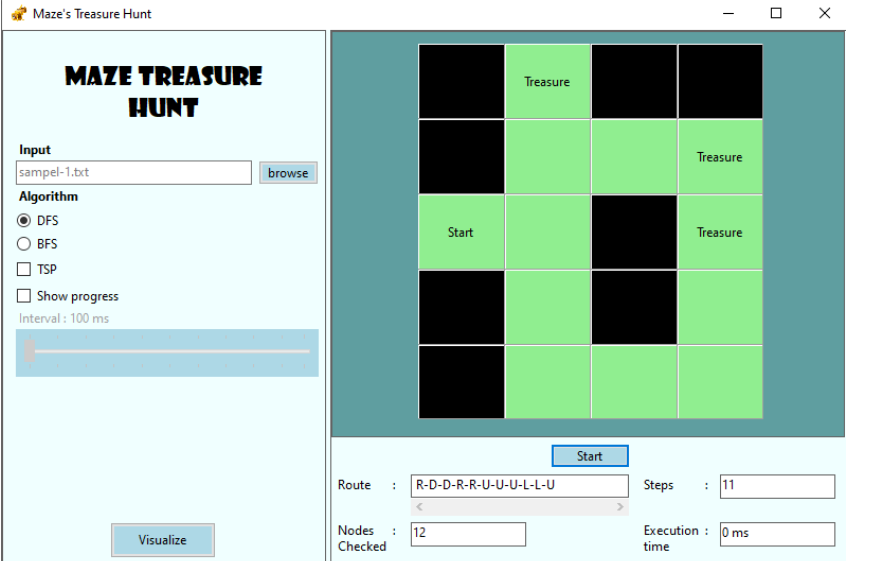
B. Mapping Persoalan menjadi Elemen Algoritma BFS dan DFS

Program akan menerima *inputan* berupa file *case Maze Treasure Hunt* atau dengan *textfield* sehingga pengguna dapat mengetik nama file *maze* yang tersedia, apabila filename tersebut ada, program akan melakukan validasi dari file *input* tersebut. Validasi dilakukan dengan memeriksa apakah tiap komponen *input* hanya berupa K, T, R, X. Apabila validasi gagal, program akan memunculkan pesan bahwa file tidak valid. Apabila validasi berhasil, program akan menampilkan visualisasi awal dari *maze treasure hunt*.


Pengguna memilih algoritma yang digunakan menggunakan *toggle* yang tersedia. Program kemudian dapat menampilkan visualisasi akhir dari *maze* (dengan pewarnaan rute solusi). Program menampilkan luaran berupa durasi eksekusi, rute solusi, banyaknya langkah, serta banyaknya node yang diperiksa. Pada program ini kelompok kami membuat prioritas pemilihan cabang yaitu secara clockwise yang mana prioritas pertama adalah cabang kanan, dilanjut dengan cabang bawah, kemudian cabang kiri dan terakhir cabang atas.

- a. Problem state: Menelusuri *grid* (simpul) yang mungkin dikunjungi hingga ditemukan rute solusi, baik secara melebar ataupun mendalam hingga didapatkan rute solusi yang memperoleh seluruh *treasure* pada *maze*.
- b. Initial state: Visualisasi awal dari file *maze treasure hunt*.
- c. Solution state: rute yang memperoleh seluruh *treasure* pada *maze*.
- d. State Space: durasi eksekusi, rute solusi, banyaknya langkah, serta banyaknya node yang diperiksa

C. Ilustrasi Kasus Lain

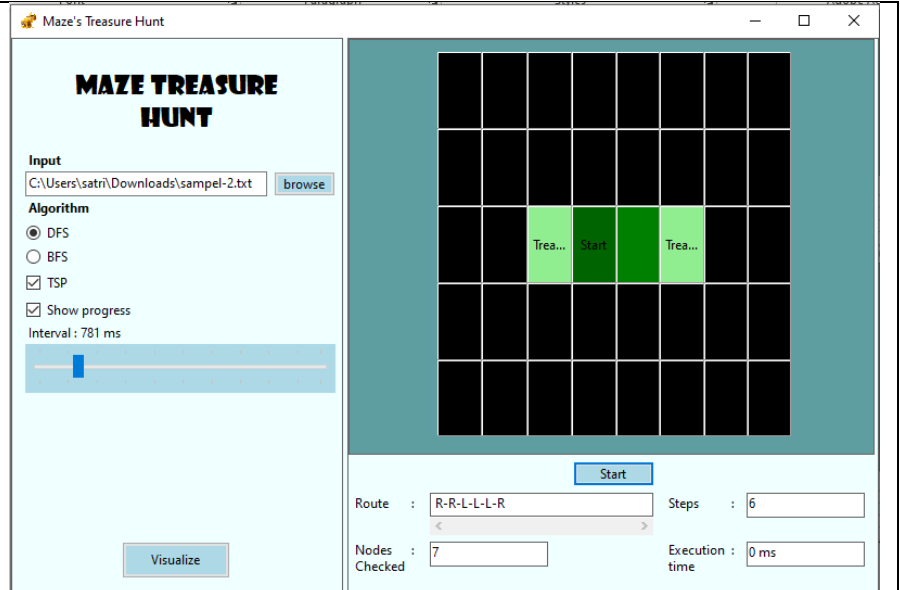
Input	Output
<p>DFS/BFS → file yang diberikan tidak valid (<i>input</i> tidak berupa K, T, R, X)</p> <p> sampel-3 - Notepad</p> <p>File Edit Format View</p> <pre> P A N G A N L U P A C E K Y A N G B E G I N I Y </pre>	
<p>DFS → <i>treasure</i> ditemukan tanpa centang <i>toggle</i> TSP dan tanpa durasi jeda setiap <i>step</i>nya</p> <p> sampel-1 - Notepad</p> <p>File Edit Format View</p> <pre> X T X X X R R T K R X T X R X R X R R R </pre>	

DFS → *treasure* ditemukan dan memberikan rute dengan menggunakan TSP serta durasi jeda setiap *step*nya sebesar 781 ms


 sampel-2 - Notepad

File Edit Format View

```
X X X X X X X X
X X X X X X X X
X X T K R T X X
X X X X X X X X
X X X X X X X X
```

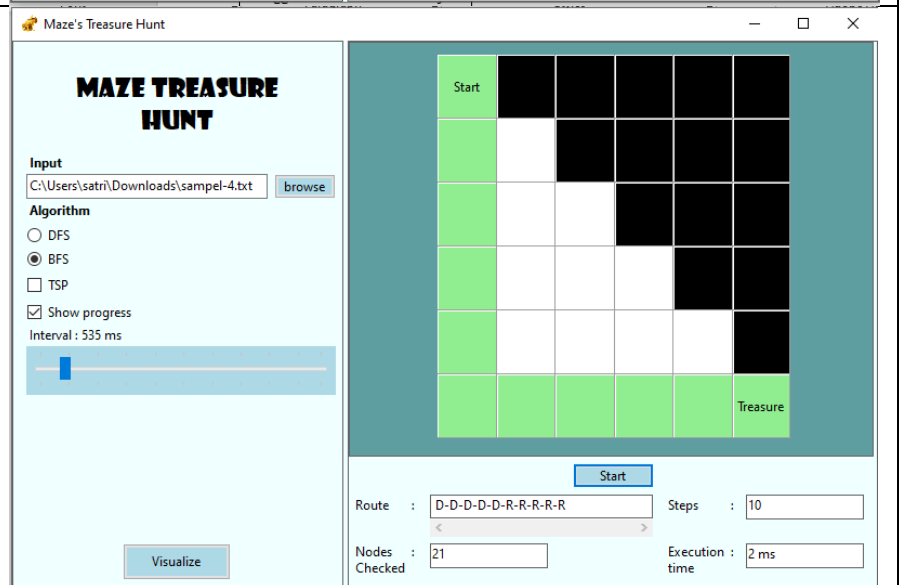


BFS → *treasure* ditemukan tanpa memanfaatkan *toggle* TSP dengan durasi jeda setiap *step*nya selama 535 ms.


 sampel-4 - Notepad

File Edit Format View

```
K X X X X X
R R X X X X
R R R X X X
R R R R X X
R R R R R X
R R R R R T
```

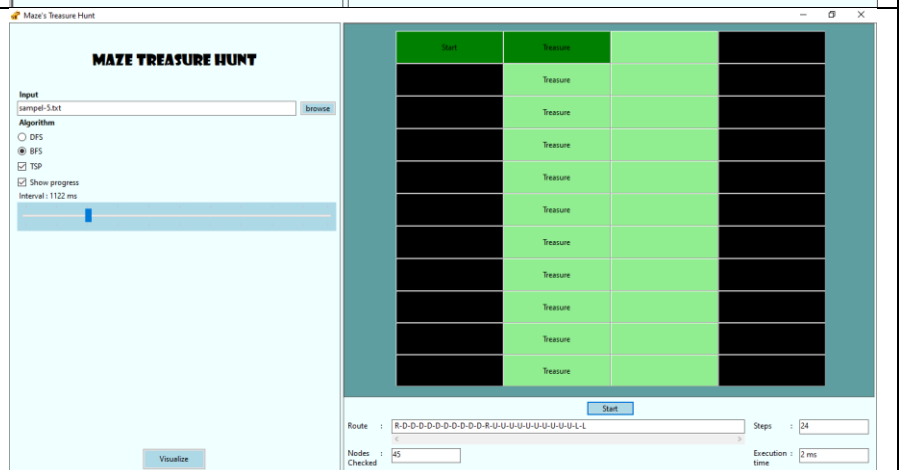


BFS → *treasure* ditemukan dengan memanfaatkan *toggle* TSP dan durasi jeda setiap *step*nya selama 1122 ms.

 sampel-5 -

File Edit Fo

```
K T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
```



BAB IV

Implementasi dan Pengujian

A. Link Repository Github

https://github.com/satrianababan/Tubes2_MazeTreasureHunt

B. Link Video Youtube

<https://youtu.be/pQM8zJmGiTU>

C. Implementasi Program (Pseudocode)

Berikut ini merupakan Pseudocode dari algoritma program kami :

BFS
<pre>function bfs(map, start, treasure) goback ← false rute ← [] visited ← {} queue ← {} prevstart ← start queue enqueue start while (queue not empty) now ← queue.dequeue() if (now is visited) continue visited ← visited ∪ now if (now is treasure) remove now from treasure queue clear if (treasure empty) if (want TSP) goback ← true else ambil subrute dari prevstart ke now rute ← rute + subrute berhenti else if (goback and now is start) ambil subrute dari prevstart ke now rute ← rute + subrute berhenti if (queue empty and tetangga now sudah divisit semua)</pre>

```

    ambil subrute dari prevstart ke now
    empty visited
    prevStart  $\leftarrow$  now

    foreach next as tetangga dari now
        if next not visited and valid cell
            previous_cell[next]  $\leftarrow$  now
            queue.enqueue(next)
    return rute

```

DFS

```

function dfs(map, start, treasure, isTSP)
    rute  $\leftarrow$  []
    visited  $\leftarrow$  {}
    stack  $\leftarrow$  {}
    stack push start
    visited  $\leftarrow$  visited  $\cup$  start

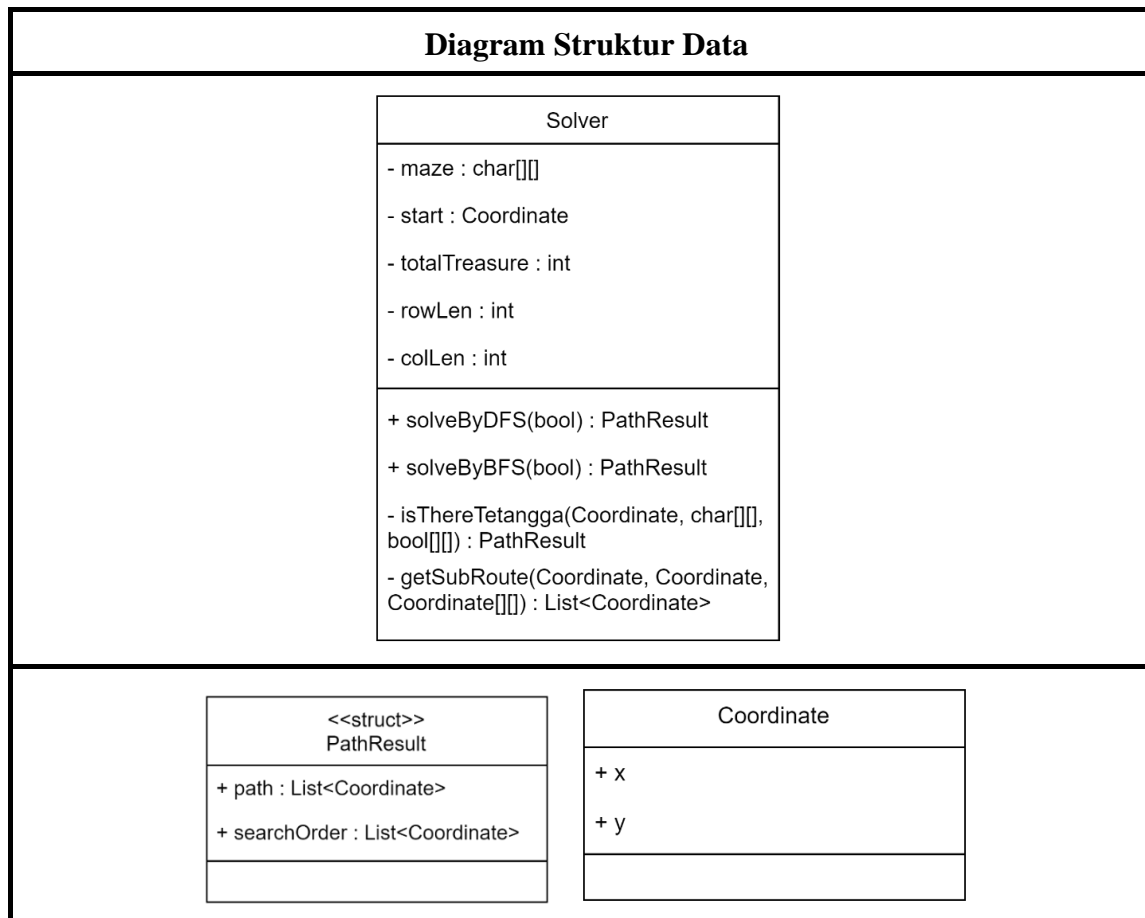
    while (stack not empty)
        now  $\leftarrow$  stack top
        rute  $\leftarrow$  rute + now
        if now is treasure then
            remove now from treasure
        end if

        if treasure is not empty then
            foreach next as tetangga dari now
                if next is not visited and next is valid
                    stack push next
                    visited  $\leftarrow$  visited  $\cup$  next
                    break
            end if
        end foreach
        if now tidak memiliki tetangga
            stack pop {lakukan backtracking}
        end if
    else
        if isTPS then
            stack pop {lakukan backtracking hingga kembali ke start}
        else
            break
        endf if
    end if
end while

return rute

```

D. Struktur Data dan Spesifikasi Program



Terdapat beberapa struktur data yang digunakan pada program, yaitu :

1. Stack

Stack merupakan struktur data yang menggunakan prinsip LIFO (*Last In First Out*) dimana penggunaannya menggunakan fungsi push (untuk memasukkan elemen) dan pop (untuk mengeluarkan elemen). Struktur data ini diimplementasikan pada algoritma DFS.

2. Queue

Queue merupakan struktur data yang menggunakan prinsip FIFO (*First In First Out*) dimana penggunaannya menggunakan fungsi enqueue (untuk memasukkan elemen) dan dequeue (untuk mengeluarkan elemen). Struktur data ini diimplementasikan pada algoritma BFS.

3. List

List merupakan struktur data yang menyimpan elemen atau objek dalam bentuk list. Terdapat 2 cara untuk menambahkan elemen ke dalam list, yaitu Add dan AddRange, dimana Add digunakan untuk menambahkan satu elemen dan AddRange digunakan untuk menambahkan beberapa elemen secara bersamaan ke dalam List. Pada program ini list dipakai untuk menyimpan koordinat rute dan koordinat urutan pencarian.

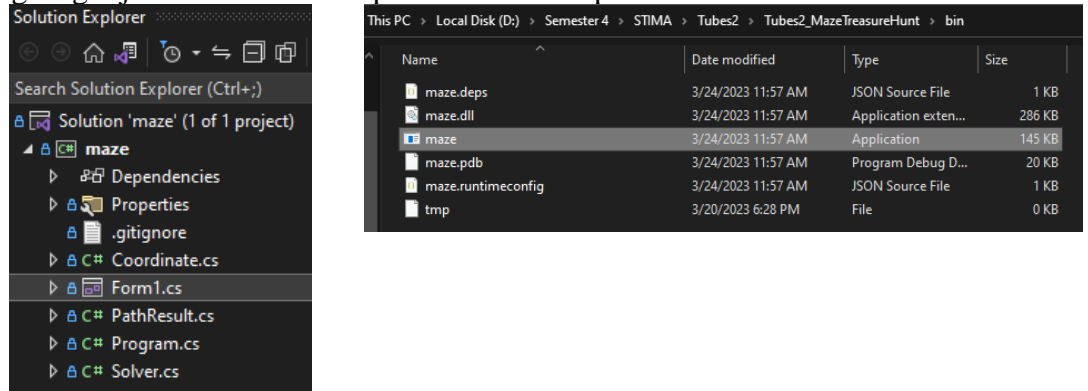
4. Array

Array adalah struktur data yang menyimpan beberapa elemen dengan tipe yang sama dalam suatu lokasi memori berdekatan. Penggunaan array ini dilakukan untuk menyimpan hasil pencarian BFS dan juga DFS. Pada program ini menggunakan array 2 dimensi untuk menyimpan data peta *maze*.

E. Tata Cara Penggunaan Program

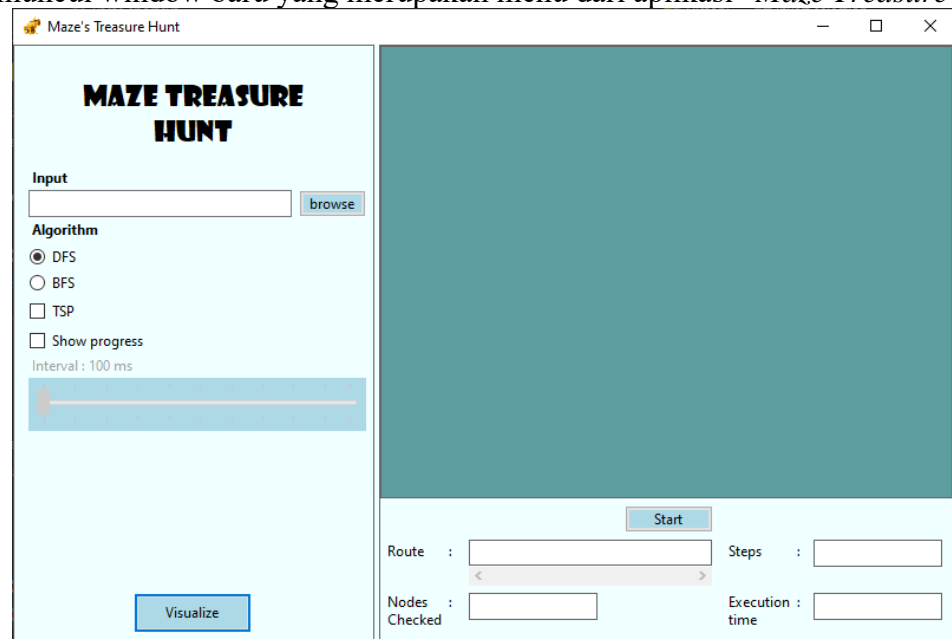
Berikut merupakan cara menggunakan program dari kelompok kami :

- Buka Visual Studio lalu lakukan clone pada repositori ini. Kemudian, pada folder src buka file solution maze.sln, dan lakukan running pada file Form1.cs. Program juga dapat langsung dijalankan melalui Aplikasi maze.exe pada folder bin.



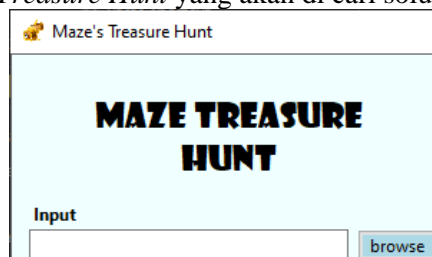
Gambar 4.1 Aplikasi maze pada folder bin dan file Form1.cs pada Slution Explorer

- Akan muncul window baru yang merupakan menu dari aplikasi “Maze Treasure Hunt”



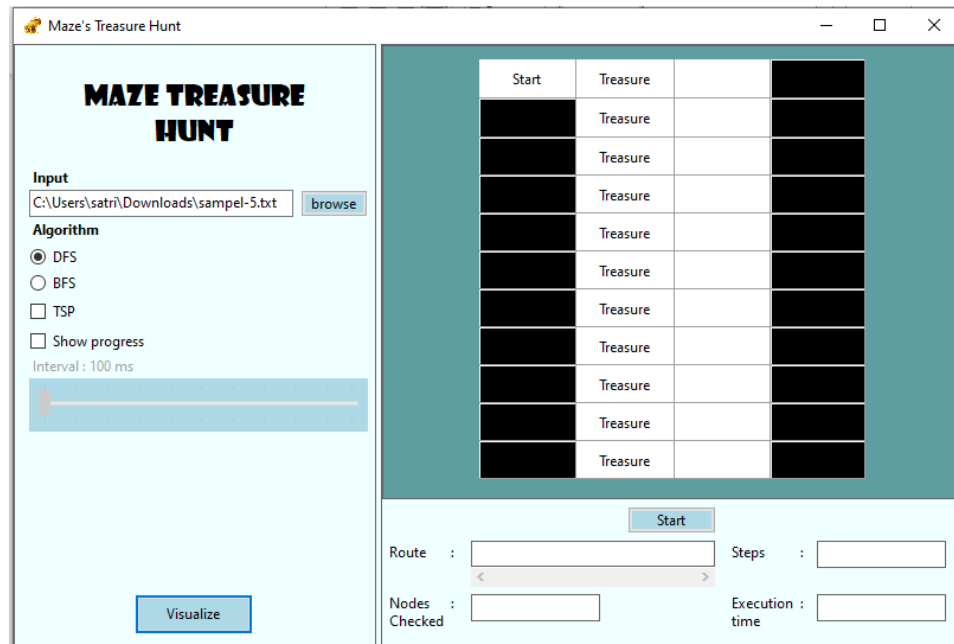
Gambar 4.2 Tampilan awal aplikasi Maze Treasure Hunt

- Pada menu *input*, dapat melakukan browse file pada local directory atau langsung mengetik nama dari file case Maze Treasure Hunt yang akan di cari solusinya.



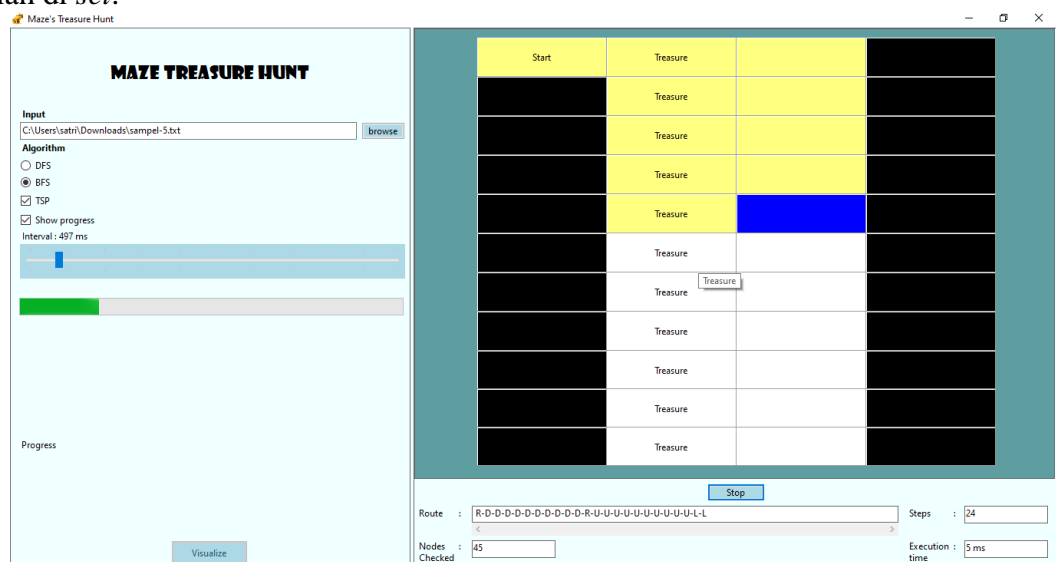
Gambar 4.3 Tampilan aplikasi untuk memilih file case Maze Treasure Hunt

- d. Setelah melakukan *input* file case *maze treasure hunt*, maka dapat langsung melihat visualisasi awal dari file case dengan mengklik tombol "Visualize". Pada tahap ini juga dapat memilih algoritma yang akan dipakai untuk menemukan *treasure*, serta memilih untuk menggunakan *toggle* TSP dan mengatur durasi jeda setiap *grid* langkah penyelesaian. Jika *set up* yang diinginkan sudah di set, dapat langsung klik tombol "Start".



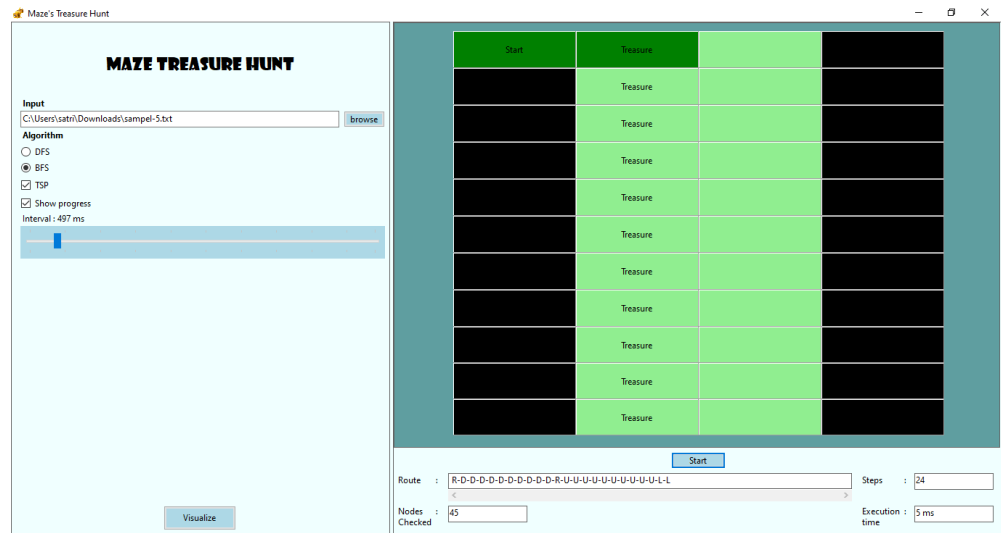
Gambar 4.4 Tampilan set up penggunaan aplikasi pada sampel-5.txt

- e. Setelah menekan tombol "Start", program akan mulai melakukan pencarian *treasure* sesuai dengan algoritma yang dipilih, serta penggunaan TSP dan durasi jeda *step* yang telah di *set*.



Gambar 4.5 Tampilan contoh proses pencarian *treasure* dengan algoritma BFS dan memanfaatkan TSP dan durasi jeda setiap *step* selama 497 ms.

- f. Selanjutnya program akan memberikan luaran berupa rute solusi dari seluruh *treasure* yang dicari, durasi eksekusi, banyaknya langkah, serta banyaknya node yang diperiksa



Gambar 4.6 Tampilan hasil pencarian treasure dengan algoritma BFS dan memanfaatkan TSP dan durasi jeda setiap step selama 497 ms.

- g. Jika sudah selesai menggunakan aplikasi, klik tombol “X” pada pojok kanan atas aplikasi untuk keluar dan menutup aplikasi.

F. Fitur-Fitur Yang Tersedia Pada Aplikasi

- *input* file case *maze treasure hunt* yang dapat langsung dibrowse dari directory local ataupun langsung mengetikkan nama file .txt yang tersedia di folder test.
- Algoritma pencarian *treasure* yang dapat menggunakan skema melebar (BFS) atau mendalam (DFS).
- *Toggle* TSP yang dapat memberikan rute solusi yang diperoleh dengan kembali ke titik awal setelah menemukan segala harta karunnya (Tetap dengan algoritma BFS atau DFS).
- *Toggle* “Show Progres”, yang memungkinkan pengguna mengatur jeda durasi setiap proses *grid step*.
- Tombol “Visualize”, yang akan menampilkan visualisasi awal dari file case yang diinputkan.
- Ketika klik tombol “Start”, program akan mulai mencari *treasure* dan memberikan luaran berupa rute solusi dari seluruh *treasure* yang dicari, durasi eksekusi, banyaknya langkah, serta banyaknya node yang diperiksa.

Penjelasan warna rute program :

- Hijau muda : *grid* hanya dikunjungi satu kali pada rute
- Hijau : *grid* dikunjungi 2 kali pada rute
- Hijau tua : *grid* dikunjungi lebih dari 2 kali

Penjelasan Warna Grid Jika memanfaatkan fitur “Show Progress” :

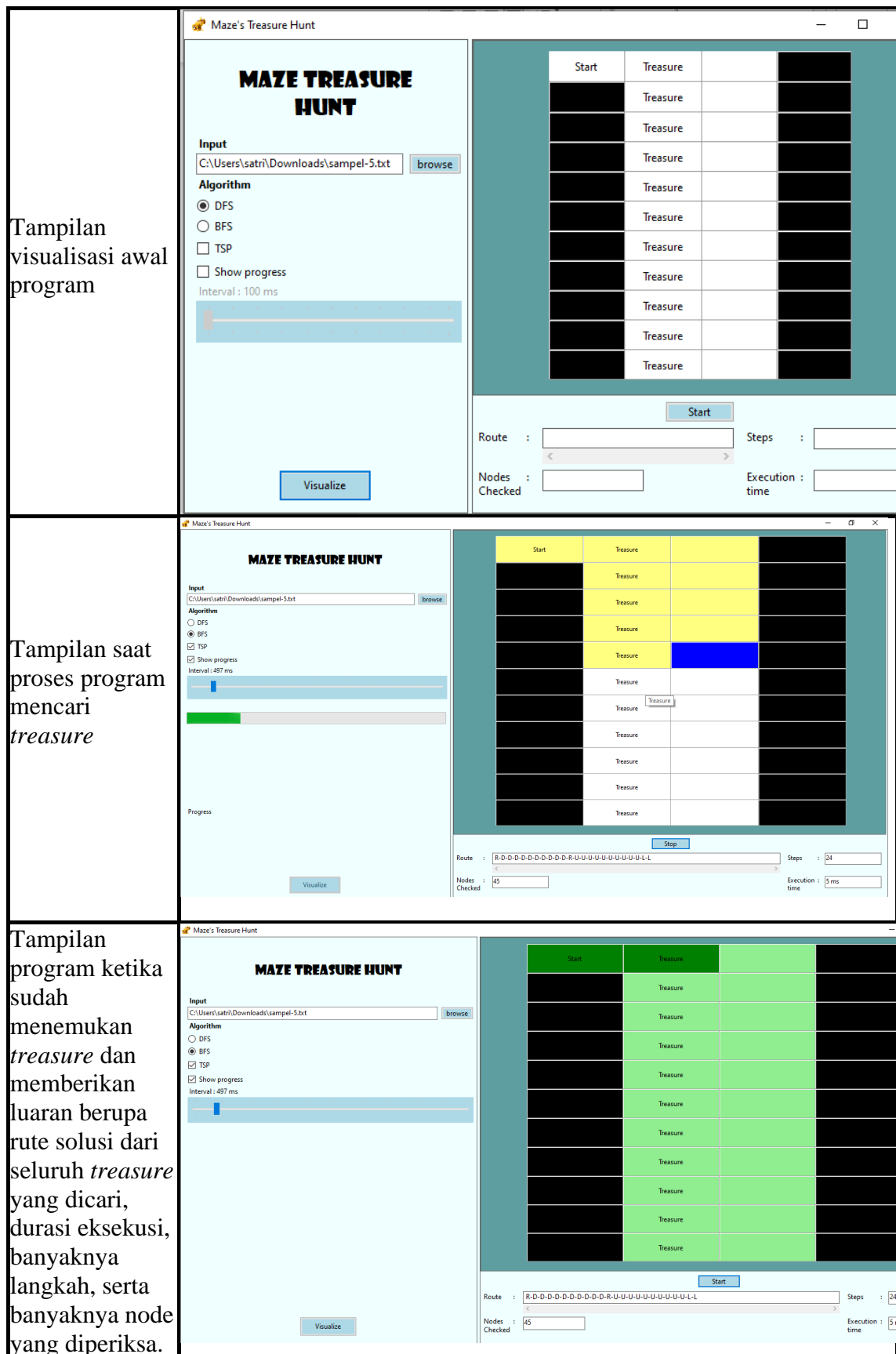
- Kuning muda : *grid* telah diproses sekali pada proses pencarian *treasure*
- Kuning : *grid* telah diproses dua kali pada proses pencarian *treasure*
- Kuning tua : *grid* telah diproses sekali pada proses pencarian *treasure*
- Biru : *grid* sedang diproses dalam pencarian *treasure*

G. Hasil Pengujian

a. Antarmuka program

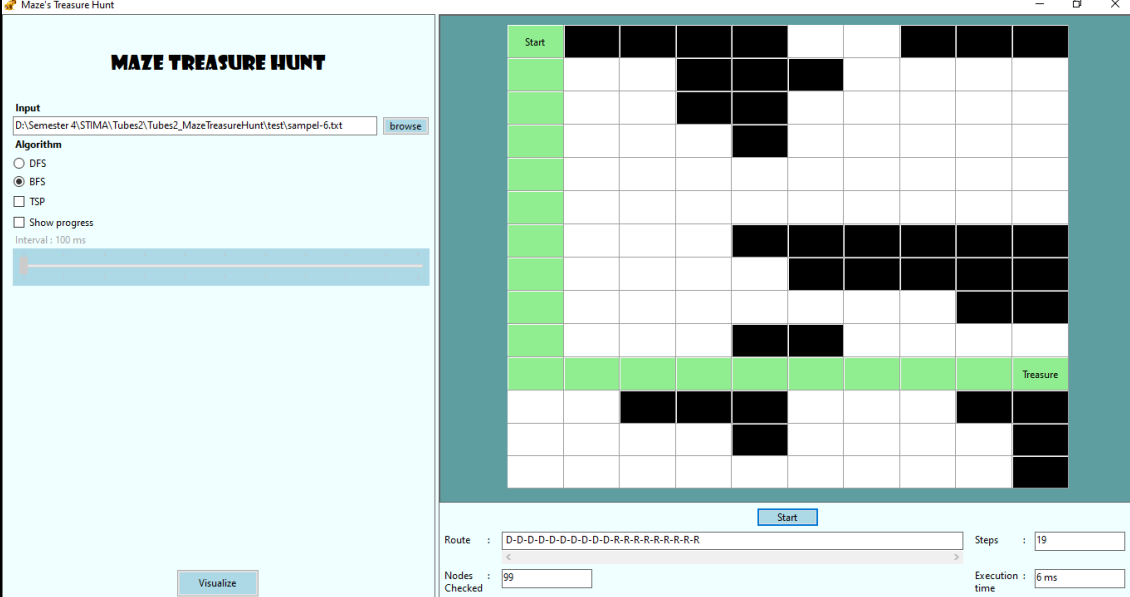
Tabel 1 Antarmuka program

Antarmuka Program	
Tampilan awal Program	
Tampilan pemilihan file <i>case</i> dari <i>directory local</i>	

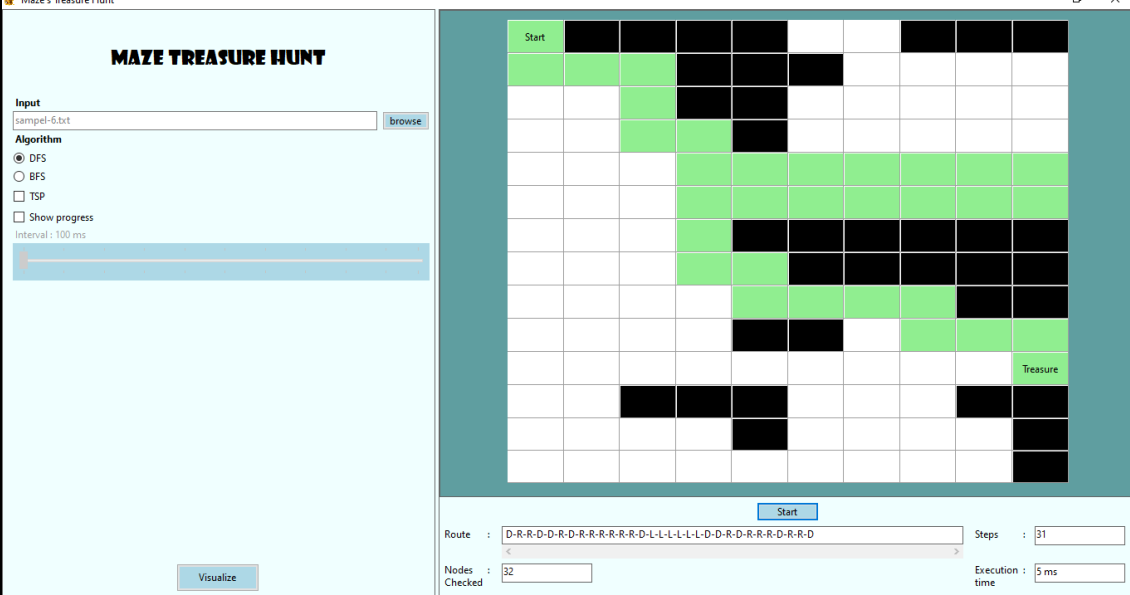


b. Data uji


Tabel 2 Data uji 1

TESTING 1	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan BFS Tidak mencentang <i>toggle</i> TSP Tidak memiliki jeda durasi setiap proses <i>grid step</i>nya.
Nama file yang dicari	sampel-6.txt
<p>Hasil Pengujian:</p> 	

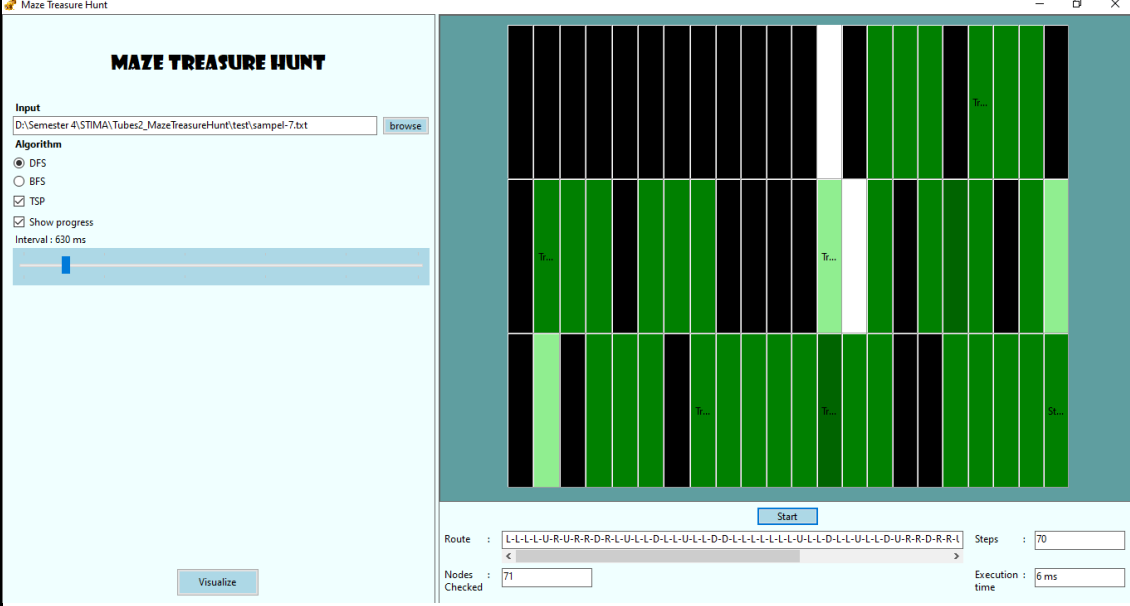
Tabel 3 Data uji 2

TESTING 2	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan DFS Tidak mencentang <i>toggle</i> TSP Tidak memiliki jeda durasi setiap proses <i>grid step</i>nya
Nama file yang dicari	sample-6.txt
<p>Hasil Pengujian :</p> 	

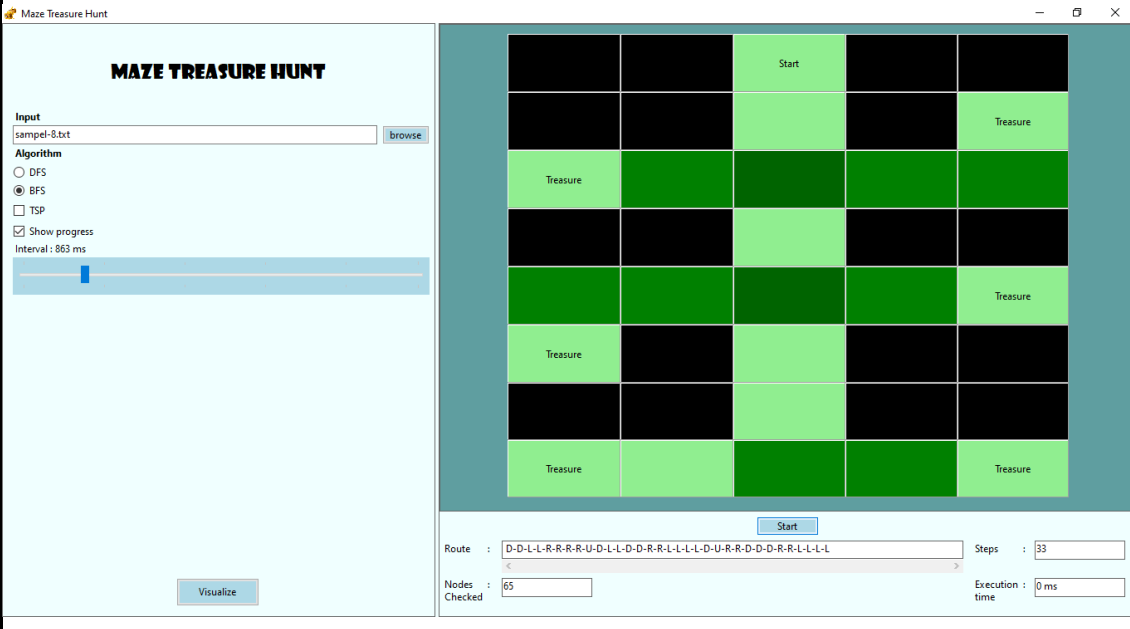
Tabel 4 Data uji 3

TESTING 3	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan BFS Mencentang <i>toggle</i> TSP Durasi jeda proses setiap <i>step grid</i> nya selama 451 ms
Nama file yang dicari	sampel-7.txt
Hasil Pengujian: 	

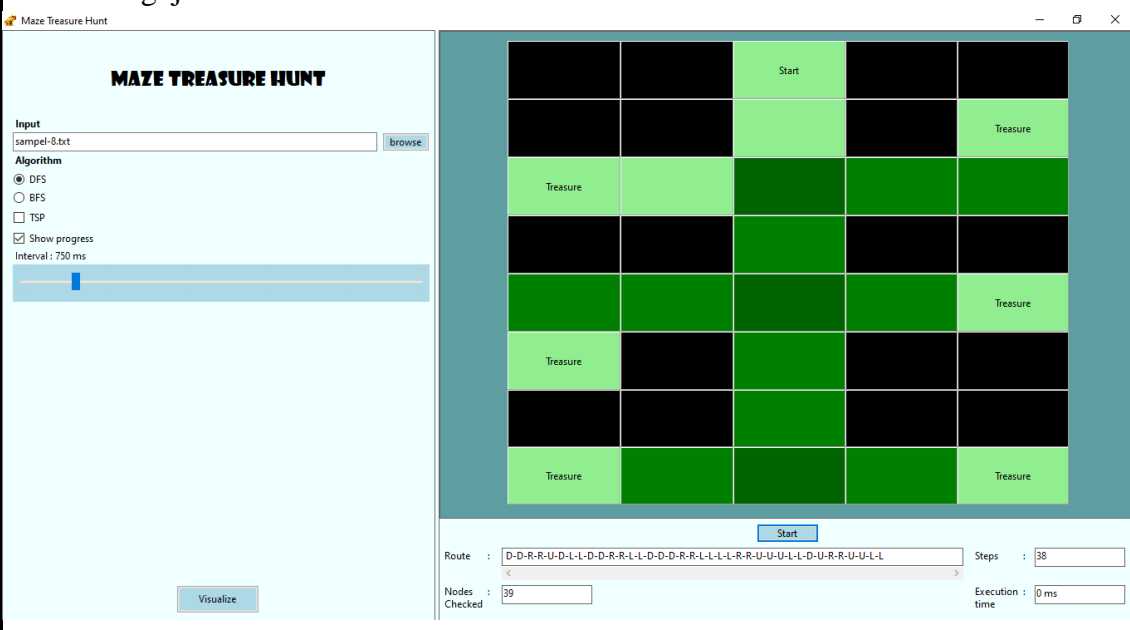
Tabel 5 Data uji 4

TESTING 4	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan DFS Mencentang <i>toggle</i> TSP Durasi jeda proses setiap <i>step grid</i> nya selama 630 ms
Nama file yang dicari	sampel-7.txt
Hasil Pengujian: 	

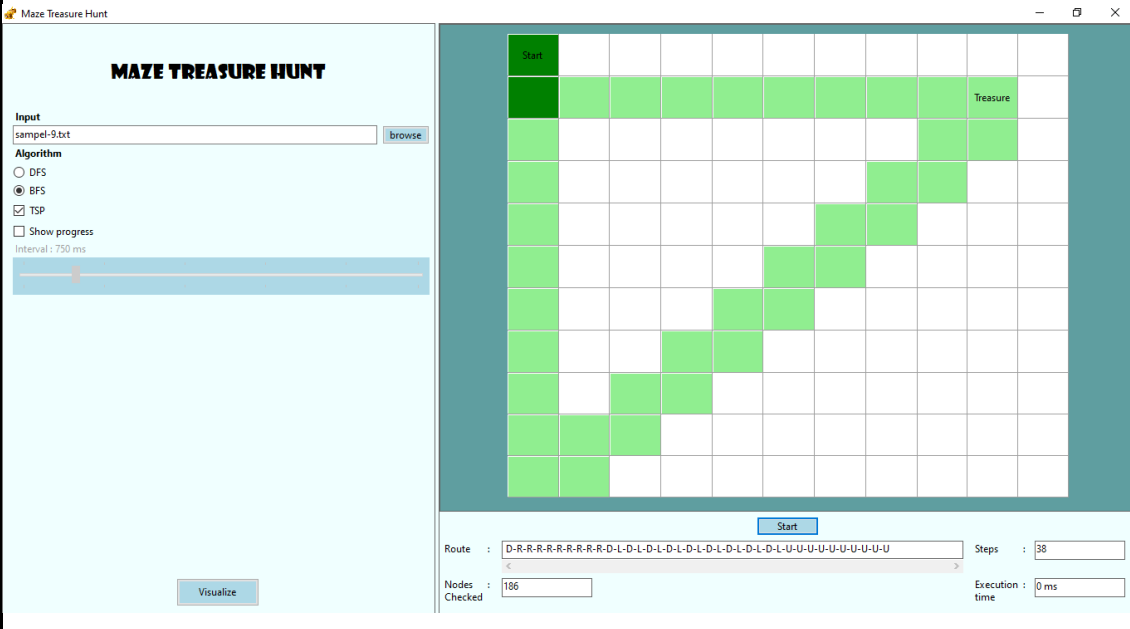
Tabel 6 Data uji 5

TESTING 5	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan BFS Tidak mencentang toggle TSP Durasi jeda proses setiap <i>step grid</i> nya selama 863 ms
Nama file yang dicari	sampel-8.txt
<p>Hasil Pengujian:</p> 	

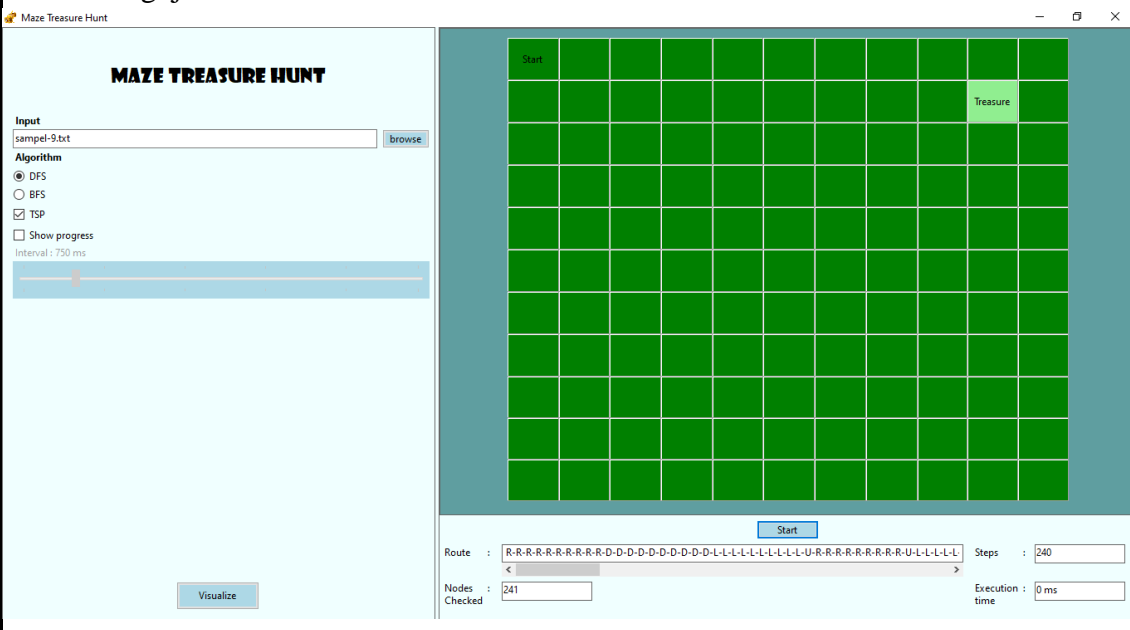
Tabel 7 Data uji 6

TESTING 6	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan BFS Tidak mencentang toggle TSP Durasi jeda proses setiap <i>step grid</i> nya selama 750 ms
Nama file yang dicari	sampel-8.txt
<p>Hasil Pengujian:</p> 	

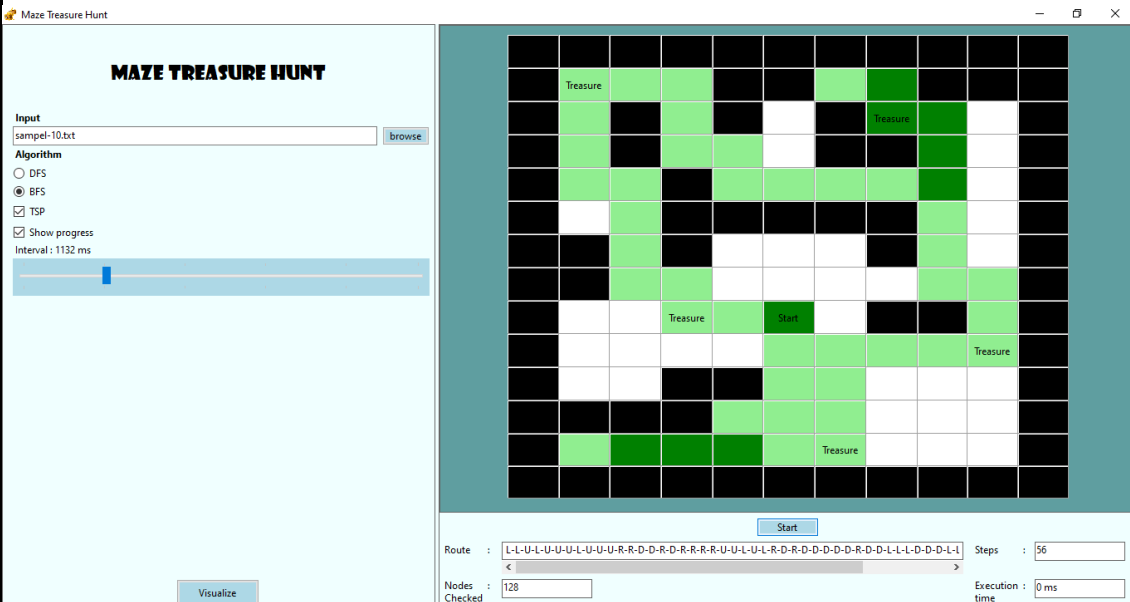
Tabel 8 Data uji 7

TESTING 7	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan DFS Mencentang toggle TSP Tanpa durasi jeda proses setiap <i>step grid</i> nya
Nama file yang dicari	sampel-9.txt
Hasil Pengujian: 	

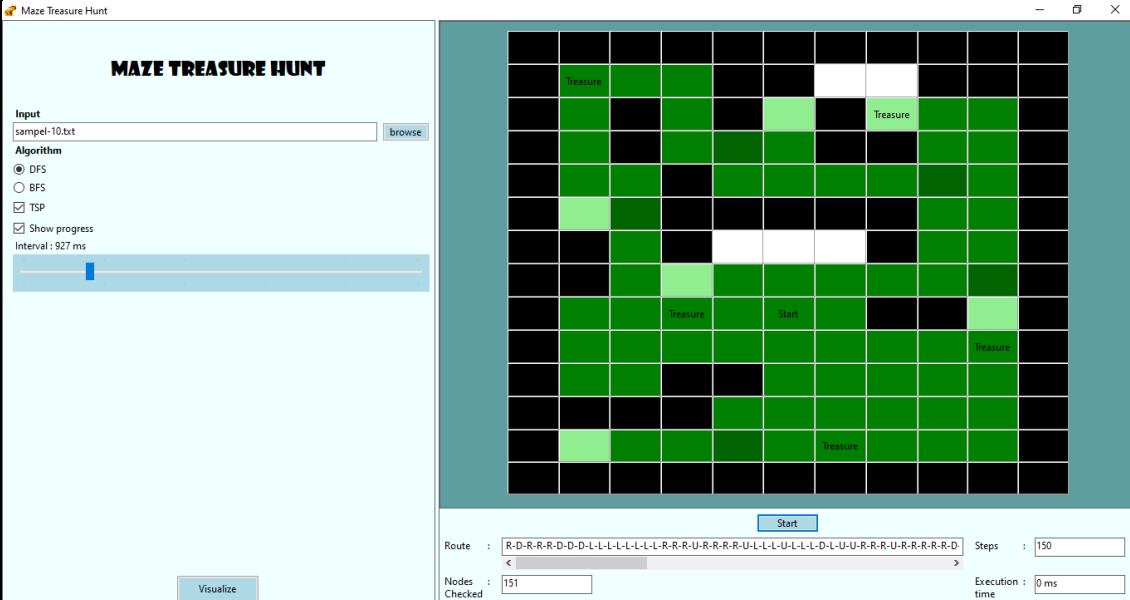
Tabel 9 Data uji 8

TESTING 8	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan DFS Mencentang toggle TSP Tanpa durasi proses setiap <i>step grid</i> nya
Nama file yang dicari	sampel-9.txt
Hasil Pengujian: 	

Tabel 10 Data uji 9

TESTING 9	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan BFS Mencentang toggle TSP Durasi jeda proses setiap <i>step grid</i> nya selama 1132 ms
Nama file yang dicari	sampel-10.txt
Hasil Pengujian: 	

Tabel 11 Data uji 10

TESTING 10	
Skenario Pengujian	<ul style="list-style-type: none"> Menggunakan DFS Mencentang toggle TSP Durasi jeda proses setiap <i>step grid</i> nya selama 927 ms
Nama file yang dicari	sampel-10.txt
Hasil Pengujian: 	

H. Analisis Desain Solusi

DFS (*Depth-First Search*) dan BFS (*Breadth-First Search*) adalah dua algoritma pencarian graf yang umum digunakan dalam pemrograman dan teori graf. DFS dan BFS berbeda dalam cara mereka menjelajahi graf dan menemukan solusi. Berikut adalah perbedaan utama antara DFS dan BFS:

- a. Strategi Pencarian:
 - DFS: Memulai dari satu simpul dan mengunjungi simpul terdekat yang belum dikunjungi. DFS terus bergerak maju hingga mencapai simpul tujuan atau simpul yang tidak memiliki cabang lagi. DFS sering digunakan untuk mencari jalan yang terpendek dari suatu simpul ke simpul lainnya.
 - BFS: Memulai dari satu simpul dan mengunjungi semua simpul tetangga sebelum melanjutkan ke simpul berikutnya. BFS menelusuri semua cabang di level yang sama terlebih dahulu sebelum melanjutkan ke level berikutnya. BFS sering digunakan untuk mencari solusi yang optimal dan terpendek dalam masalah pencarian dengan banyak solusi.
- b. Struktur Data:
 - DFS: Menggunakan struktur data tumpukan (*stack*) untuk menyimpan simpul yang sudah dikunjungi dan masih harus dikunjungi.
 - BFS: Menggunakan struktur data antrian (*queue*) untuk menyimpan simpul yang sudah dikunjungi dan masih harus dikunjungi.
- c. Kompleksitas:
 - DFS: Kompleksitas waktu DFS adalah $O(V+E)$, di mana V adalah jumlah simpul dan E adalah jumlah sisi pada graf. DFS tidak membutuhkan banyak memori.
 - BFS: Kompleksitas waktu BFS juga $O(V+E)$. Namun, BFS membutuhkan lebih banyak memori daripada DFS karena harus menyimpan semua simpul yang sudah dikunjungi dalam sebuah antrian.
- d. Aplikasi:
 - DFS: DFS sering digunakan untuk mencari lintasan terpendek pada graf yang tidak berbentuk pohon, seperti graf berarah atau graf dengan siklus.
 - BFS: BFS sering digunakan untuk mencari solusi terpendek dalam masalah pencarian dengan banyak solusi, seperti dalam masalah permainan.

Dari hasil pengujian yang sudah lakukan, perbedaan antara strategi BFS dan DFS sangat terlihat pada Testing 1,3,5,7,9 (BFS) dengan testing 2,4,6,8,10. (DFS).

BAB V

Penutup

A. Kesimpulan

Berdasarkan hasil pencarian *treasure* dan analisisnya dapat disimpulkan bahwa, meskipun DFS dan BFS merupakan algoritma pencarian graf, DFS dan BFS memiliki strategi pencarian, struktur data, kompleksitas, dan aplikasi yang berbeda. Pemilihan algoritma yang tepat tergantung pada jenis masalah yang sedang diselesaikan dan karakteristik dari graf yang digunakan. Terbukti dari hasil pengujian yang mana DFS dan BFS memberikan rute solusi yang berbeda, durasi eksekusi yang berbeda, banyaknya langkah yang berbeda, serta banyaknya node yang diperiksa juga berbeda.

B. Saran

Saran yang dapat kami berikan bagi pembaca dari pengerjaan Tugas Besar 2 IF2211 Strategi Algoritma semester 2 2021/2022 adalah:

1. Melakukan eksplorasi lebih dalam lagi terkait C# Desktop Application Development. Mulai dari struktur file yang ada pada aplikasi ini, cara kerja, maupun kreatifitas dalam pembuatan *graphical user interface* (GUI).
2. Melengkapi *code program* dengan komentar, agar dalam pengembangan atau *debug* dapat memahami *code* dengan mudah.

C. Refleksi

Melalui tugas besar ini kami dapat semakin mengerti perbedaan dalam implementasi nyata penggunaan algoritma DFS dan BFS. Kedua algoritma tersebut memiliki keunggulannya masing-masing sehingga untuk mendapatkan solusi optimal perlu dilakukan analisis terkait jenis masalah yang sedang diselesaikan, sehingga kita dapat memilih algoritma yang tepat yang memberikan solusi terbaik.

D. Tanggapan Terkait Tugas Besar Ini

Tanggapan kami terkait tugas besar ini yaitu kami mendapatkan wawasan dan pengalaman baru dalam membuat project berbasis bahasa C#. Ini merupakan pengalaman baru yang semakin meningkatkan kepekaan kami dalam memahami perbedaan bahasa pemrograman yang ada sehingga kami dapat semakin eksplorasi diri terkait penggunaan bahasa pemrograman terbaik dalam menyelesaikan suatu project tertentu. Pengalaman dan pemahaman akan penggunaan *Graphical user interface* (GUI), juga semakin terlatih melalui penyelesaian tugas besar ini, dan tentunya pemahaman akan DFS dan BFS yang semakin meningkat.

Sangat terimakasih buat asisten yang telah dengan kreatif memberikan tugas besar ini, terimakasih.

Daftar Pustaka

- Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2023). “Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1)” Diakses online dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf> pada 21 Maret 2023.
- Munir, Rinaldi. (2020). “ Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 2)”. Diakses online dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf> pada 21 Maret 2023.