

LAPORAN TUGAS BESAR II

IF2211 Strategi Algoritma



Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt

Naufal Syifa Firdaus 13521050

Moch Sofyan Firdaus 13521083

Razzan Daksana Y 13521087

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022/2023

Daftar Isi

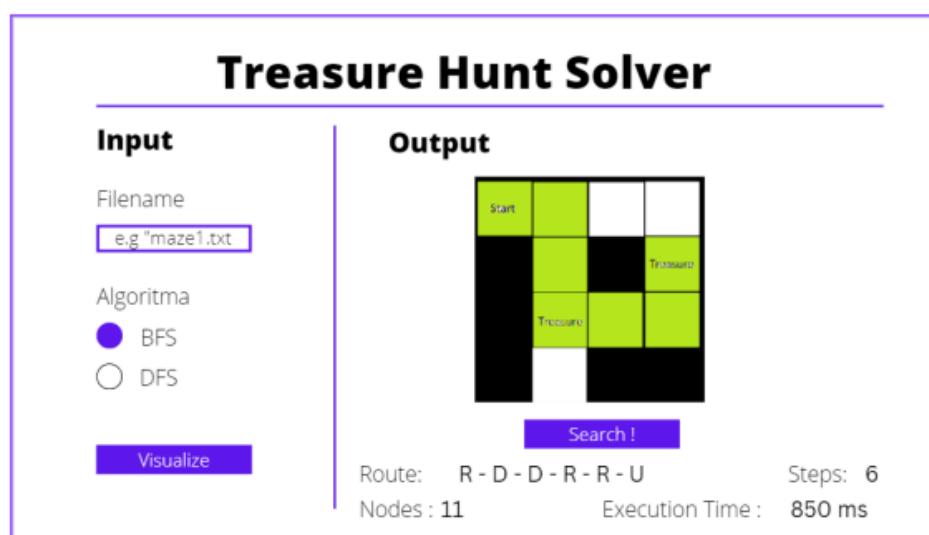
BAB I: Deskripsi Tugas	5
BAB II :Landasan Teori	6
2.1. Graph Traversal	6
2.2. Traveling Salesperson	6
2.3. Breadth First Search	7
2.4. Depth First Search	7
2.5. Avalonia UI	7
BAB III :Rancangan Implementasi	8
3.1. Pemecahan Masalah	8
3.2. Penerapan Algoritma BFS dan DFS	9
3.3. Analisis Kemungkinan Kasus	10
BAB IV : Realisasi Aplikasi Desktop	11
4.1. Menjalankan Program	11
4.2. Spesifikasi dan Struktur Data Program	12
4.3. Implementasi Program	16
4.4. Eksperimen	18
4.5. Analisis Hasil Eksperimen	38
BAB V :Kesimpulan dan Saran	39
5.1. Kesimpulan	39
5.2. Saran	39
5.3. Refleksi	39
Daftar Pustaka	40
Lampiran	41

BAB I: Deskripsi Tugas

Tugas Besar 2 Strategi Algoritma adalah 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. Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), telusuri 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.

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. Visualisasikan 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.

Berikut contoh GUI program dengan Masukan dan Keluaran yang diharapkan.



BAB II :Landasan Teori

2.1. Graph Traversal

Graf Traversal adalah proses menelusuri sebuah graf secara sistematis sedemikian rupa sehingga setiap simpul dalam graf tersebut terkunjungi. Dalam graf traversal, sebagian simpul dapat dikunjungi lebih dari satu kali, sehingga jika graf cukup kompleks dengan simpul yang banyak, penerapan algoritmanya seringkali menggunakan sistem yang menyimpan simpul yang sudah dikunjungi. Traversal Graf dapat dilakukan menggunakan banyak algoritma penelusuran graf yang menggunakan pendekatan berbeda dalam memutuskan simpul mana yang akan dikunjungi setiap langkahnya. Meskipun terdapat banyak algoritma untuk melakukan traversal graf, algoritma yang digunakan untuk menyelesaikan masalah dalam tugas besar ini adalah Breadth First Search (BFS) dan Depth First Search (DFS). DFS dan BFS termasuk kedalam implementasi traversal graf dengan kasus khusus dimana “*Parent*” dari sebuah simpul selalu diasumsikan sudah dikunjungi. Sehingga graf direpresentasikan dengan sebuah pohon.

Konsep traversal graf dapat digunakan untuk pemecahan masalah. Masalah yang dapat diselesaikan dengan konsep tersebut harus dapat diterjemahkan atau direpresentasikan dengan graf. Traversal Graf dinyatakan menghasilkan solusi jika kondisi yang ditetapkan sebagai solusi terpenuhi dengan langkah traversal, dalam hal ini ditemukan semua “*treasure*” dalam graf. Masalah yang dapat diselesaikan dengan metode ini antara lain penjelajahan web, penelusuran file direktori, penyelesaian labirin.

2.2. Traveling Salesperson

Traveling Salesperson atau TSP merupakan masalah kombinatorial untuk mencari rute perjalanan terpendek dari seorang penjual. Penjual harus mengunjungi kota-kota yang berbeda dan akhirnya kembali ke tempat awal dia memulai perjalanan. Secara umum, persoalan TSP adalah mencari terpendek diantara beberapa titik yang diberikan dimana rute tersebut melintasi setiap titik tepat satu kali dan kembali ke titik awal. Pada persoalan tugas besar kali ini TSP dimaksudkan agar pencarian kembali ke titik awal setelah mendapatkan semua harta karun.

2.3. Breadth First Search

Breadth First Search atau pencarian melebar adalah salah satu algoritma untuk menelusuri sebuah graf dengan mengutamakan mengunjungi simpul yang bertetangga (kedalamannya sama) dengan simpul yang sekarang. Misal simpul awal yang dikunjungi adalah x dalam sebuah graf. Langkah melakukan BFS adalah sebagai berikut.

1. Kunjungi semua simpul yang kedalamannya sama dengan simpul x .
2. Ketika semua simpul tetangga sudah terkunjungi, kunjungi “*child*” dari x dan kunjungi semua tetangganya.
3. Ulangi langkah diatas dengan semua tetangga x sampai semua simpul dalam graf terkunjungi.

2.4. Depth First Search

Depth First Search atau pencarian mendalam adalah metode untuk menelusuri sebuah graf dengan mengunjungi “*child*” dari simpul yang sekarang dan terus mengunjungi “*child*” simpul sampai tidak ditemukan anak simpul lagi. Untuk lebih lengkapnya lagi berikut langkah algoritma dari pencarian mendalam.

1. Kunjungi *child* dari simpul yang sekarang sedang dikunjungi.
2. Ulangi lagi langkah diatas sampai simpul yang dikunjungi tidak punya anak.
3. Kembali ke simpul terakhir yang dikunjungi yang punya tetangga dengan kedalaman sama dan langkah 1 sampai 2.
4. Ulangi semua langkah diatas hingga simpul yang mungkin dikunjungi telah habis.

2.5. Avalonia UI

Avalonia UI adalah framework untuk mengembangkan user interface aplikasi multi-platform desktop, mobile, dan web berbasis .NET. Avalonia menyediakan framework untuk membuat Graphical User Interface diatas aplikasi .NET menggunakan format XAML dengan sintaks yang mirip dengan WPF. Gratis dan open-source menjadikan Avalonia alat yang efektif untuk membuat user interface untuk aplikasi khususnya pada platform desktop dengan sistem operasi windows. Karena kapabilitasnya dalam mengostumisasi IU, dengan penggunaan

yang tepat, aplikasi yang dibuat dapat menghasilkan GUI dengan style yang sesuai dengan kemauan pengembang. Avalonia dapat mengakomodasi kreativitas perancang GUI yang sangat luas. Framework ini kompatibel dengan platform .NET 2.0 keatas dan dapat berjalan di sistem operasi Windows 8 keatas, macOS High Sierra keatas, Debian 9 keatas, Ubuntu 16.04 keatas dan Fedora 30 keatas.

BAB III :Rancangan Implementasi

3.1. Pemecahan Masalah

Sebagai permulaan, masalah yang harus diselesaikan adalah berupa masukan file .txt yang merepresentasikan sebuah labirin berisi harta karun. File tersebut kemudian kami terjemahkan ke dalam bentuk graf dengan sisi dan simpul yang sesuai dengan labirin tersebut. Hasil terjemahan tersebut disimpan dalam suatu struktur data atau kelas sehingga dapat diproses dengan algoritma penelusuran. Maka dari itu dalam membuat program DoraTheExplorer kami menggunakan paradigma pemrograman objek yang dinilai dapat merepresentasikan sebuah graf dan komponen-komponen didalamnya dengan baik. Kelas yang dibuat merupakan hasil terjemahan beserta tipe objek yang akan membantu keberjalanannya algoritma dan akan dibahas lebih lanjut pada bab berikutnya.

Secara garis besar, tiap karakter pada file .txt adalah sebuah *cell* dalam matriks dan jika *cell* tersebut dapat dikunjungi maka karakter tersebut juga adalah sebuah simpul dalam graf. Simpul yang satu dengan yang lainnya dihubungkan dengan sebuah sisi. Sisi dalam cakupan permasalahan ini mempunyai empat jenis, atas, bawah, kanan, dan kiri. Misalkan sebuah simpul mempunyai simpul tetangga di sebelah kiri dalam matriks, maka simpul tersebut akan dihubungkan dengan sisi kiri.

Grafis labirin dan solusinya, setelah diterapkan algoritma, juga perlu untuk ditampilkan dalam GUI. Avalonia menyediakan kontrol yang bernama StackPanel untuk menyusun grid yang membentuk labirin. Dengan fitur ini kami menginisiasi objek StackPanel dan mengisinya dengan Panel yaitu komponen yang dapat merender elemen didalamnya. Panel dalam konteks ini kami gunakan sebagai visualisasi satu *cell* dalam matriks labirin. Untuk menggambarkan *cell* yang dapat dikunjungi, tidak dapat dikunjungi, belum dikunjungi, sudah dikunjungi, sedang dikunjungi, dan mengandung harta karun, digunakan gambar ilustrasi yang berbeda-beda.

Terakhir, setiap langkah yang diambil oleh algoritma baik DFS maupun BFS direkam dalam sebuah larik yang berisi *state*. Sebuah *state* menyimpan bentuk labirin dan kondisi tiap *cell* di dalamnya (sudah, belum, dan sedang dikunjungi). Pada saat pemutaran animasi *state* akan menjadi sumber data visualisasi yang dapat menggambarkan proses algoritma secara keseluruhan. Kami juga mengimplementasikan *counter* untuk jumlah langkah algoritma dan simpul yang dikunjungi serta *timer* untuk waktu eksekusi algoritma.

3.2. Penerapan Algoritma BFS dan DFS

Algoritma BFS dan DFS pada dasarnya membutuhkan graf pohon sebagai representasi masalahnya. Dengan pemaparan pada sub bab sebelumnya, sudah dijelaskan garis besar mengenai menerjemahkan masukan file .txt ke kelas representasi labirin dalam program. Dengan begitu data labirin yang dihasilkan dapat diolah oleh algoritma BFS dan DFS sesuai caranya masing-masing. Walaupun langkah algoritma yang digunakan berbeda, cara tiap algoritma menangani data labirin pada dasarnya sama. Algoritma program menerima informasi berupa graf dan *state* awal dari program. *State* awal adalah situasi awal dimana hanya ada satu *node* yang sedang dikunjungi dan semua *node* lain berada dalam kondisi belum dikunjungi. Setiap iterasi langkah yang diambil algoritma menghasilkan *state* baru yang menyimpan informasi *node-node* yang telah dikunjungi. *State* tersebut kemudian disimpan dalam sebuah larik. Fungsi algoritma DFS dan BFS dalam program masing-masing menghasilkan larik koordinat dan *state* yang adalah solusi dari permasalahan yang diberikan.

Selama keberjalanan algoritma BFS dan DFS, semua *node* yang telah dikunjungi direkam dalam atribut dari *state* yang berupa larik. Hal ini bertujuan agar setiap *node* tidak

BFS dan DFS yang diimplementasikan dalam program berdasar pada definisi masing-masing algoritma yang sudah dipaparkan pada bab 2. Proses penelusuran algoritma ditampilkan dalam bentuk menyerupai pemutaran video dengan fitur yang mirip pula (play, pause, dan seeking).

3.3. Analisis Kemungkinan Kasus

Pada sebuah permasalahan tentu terdapat beberapa kasus unik yang memiliki resiko tidak tertangani oleh algoritma program. Dalam konteks tugas besar ini, file test case yang diberikan akan mengandung kasus unik yang menguji batasan algoritma program. Berikut contoh kemungkinan kasus unik dan upaya dalam menanganinya.

1. Ukuran Labirin yang Besar

Dalam kasus ini diharapkan daya komputasi dari komputer yang menjalankan program sudah memadai untuk menangani kasus labirin yang besar.

2. Labirin dengan hanya satu *node* yang dapat dikunjungi

File text dianggap sebagai invalid dan tidak bisa menjalankan pencarian oleh program karena tidak ada harta karun sehingga kondisi pemecahan masalah tidak dapat terpenuhi.

3. File bukan dalam format .txt

File tidak akan bisa dimasukan kedalam program karena program akan meminta file dengan format spesifik yaitu .txt

4. Treasure belum ditemukan tetapi tidak *node* terblokir oleh dinding atau *node* yang telah dikunjungi

Untuk menangani kasus ini kami menghapus data semua *node* yang telah dikunjungi sehingga *node-node* tersebut dapat dikunjungi lagi. Data *node* yang dihapus disimpan dalam larik lain untuk tujuan visualisasi.

5. Terdapat harta karun yang tidak dapat dijangkau

Program akan mencoba mencari harta karun tersebut tetapi pada akhirnya akan menyerah dan hanya menunjukkan jalur ke semua harta karun yang tercapai.

Akan tetapi, program akan *crash* jika algoritma TSP diminta karena algoritma tersebut mengasumsikan semua harta karun yang diminta telah tercapai.

BAB IV : Realisasi Aplikasi Desktop

4.1. Menjalankan Program

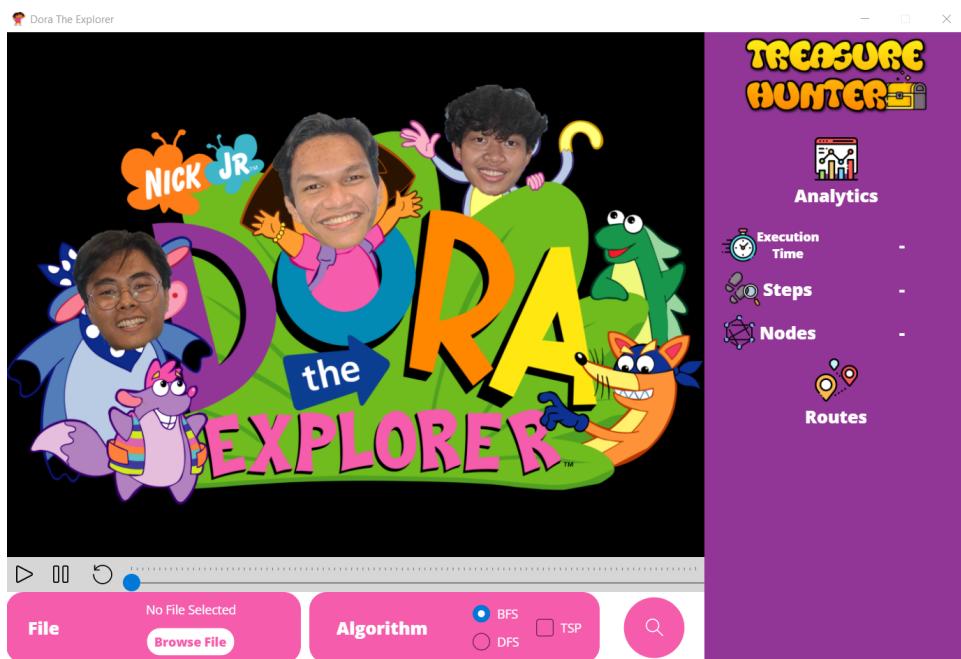
Program dapat dijalankan dengan kebutuhan sistem yang disarankan sebagai berikut:

Platform : .NET v7.0

Sistem Operasi: Windows 10,11 dan Ubuntu 22.04

Kode program dapat ditinjau dengan menggunakan IDE Visual Studio atau JetBrains Rider. Untuk Visual Studio disarankan untuk menginstall *extension* Avalonia sedangkan untuk Rider dapat menginstall plugin untuk Avalonia. Informasi lebih lengkap untuk cara install Avalonia dapat dilihat pada [website resmi AvaloniaUI](#). Aplikasi desktop DoraTheExplorer dapat dijalankan dengan terlebih dahulu *clone* GitHub repository di link pada lampiran. Dan menjalankan command **dotnet run** pada terminal dengan directory repository tersebut.

4.2. Spesifikasi dan Struktur Data Program



DoraTheExplorer menggunakan GUI sederhana untuk membantu *input* file dan juga sebagai media presentasi algoritma. Terdapat tombol yang dapat diklik dan akan menampilkan window untuk memilih file .txt labirin. *Radio button* dan *check mark* digunakan untuk menentukan algoritma apa yang akan dipakai untuk menyelesaikan persoalan. Disamping ada

kolom untuk menampilkan data pendukung seperti waktu eksekusi, jumlah langkah, simpul yang dikunjungi dan rute yang diambil. Bagian tengah dimanfaatkan untuk visualisasi labirin dengan slider yang berguna untuk *seek* proses algoritma yang ditampilkan.

Seperti yang sudah dibahas sebelumnya, program menggunakan paradigma pemrograman objek dengan kelas-kelas yang merepresentasikan objek dalam lingkup persoalan. Berikut adalah daftar kelas yang didefinisikan dalam program.

1. Cell

Cell	
Deskripsi	Kelas Cell merepresentasikan sebuah cell matriks labirin.
Atribut	
Coord : Coordinate	Menyimpan lokasi Cell dalam matriks dengan koordinat x dan y.
Visitable : Boolean	Menyatakan sebuah Cell dapat dikunjungi atau tidak.

2. CompressedState

CompressedState	
Deskripsi	Kelas berikut merepresentasikan suatu langkah dalam proses pencarian menggunakan DFS atau BFS.
Atribut	
CurrentLocation : Coordinate	Menyimpan koordinat Cell yang sedang dikunjungi.
Dir : Direction	
_visitedLocations : integer array	Menyimpan Cell yang sudah dikunjungi
_savedVisitedLocations : integer array	Menyimpan Cell yang sudah dikunjungi dan memenuhi kriteria khusus.
_width : integer	Lebar labirin

<code>_height : integer</code>	Tinggi Labirin
Method	
AddVisitedLocation : void	Menambah Cell yang sudah dikunjungi
AddSavedVisitedLocation : void	Menambah Cell yang sudah dikunjungi dan memenuhi kriteria khusus.
IsVisited : boolean	Menghasilkan true jika sebuah koordinat sudah dikunjungi.
RemoveVisitedLocation : void	Menghapus Cell yang sudah dikunjungi sesuai dengan masukan koordinat dan pindahkan ke <code>_savedVisistedLocations</code>
IsSavedVisited : boolean	Menghasilkan true jika sebuah koordinat sudah dikunjungi dalam atribut <code>_savedVisistedLocations</code>
Calculate : integer	Menghitung indeks dan lokasi bit dari Cell dalam <code>_visitedLocations</code> dan <code>_savedVisistedLocations</code>

3. Coordinate

Coordinate	
Deskripsi	Merepresentasikan sebuah koordinat dalam bidang 2 dimensi menggunakan sumbu x dan y.
Atribut	
X : integer	Sumbu X
Y : integer	Sumbu Y
Method	
Equals : boolean	Menghasilkan true jika dua koordinat sama.
ToString : string	Mengkonversi koordinat ke dalam bentuk string.
GetHashCode : integer	Menghasilkan kode hash dari koordinat.

4. Graph

Graph	
Deskripsi	Bentuk graf dalam kelas.
Atribut	
_vertices : LinkedList<Vertex>	Kumpulan simpul dalam graf.
Method	
AddVertex : void	Menambah simpul.

5. SolutionMatrix

SolutionMatrix	
Deskripsi	Kelas berikut menyimpan semua informasi yang diinisiasi dalam program termasuk state dan lokasi harta.
Atribut	
Width : integer	Lebar labirin
Height : integer	Tinggi Labirin
Path : List<Coordinate>	Jalur solusi yang dilewati
Cells : List<Cell>	Kumpulan Cell dalam matriks.
States : List<CompressedState>	Kumpulan state yang telah dilakukan.
TreasureLocations : List<Coordinate>	Lokasi semua harta.
Method	

AddState : void	Menambah state
AddTreasureLocation : void	Menambah lokasi harta
AddCell : void	Menambah Cell.
SetStates : void	Mendefinisikan list state.
SetPath : void	Mendefinisikan list jalur

6. Vertex

Vertex	
Deskripsi	Mendefinisikan simpul dalam graf beserta sisinya.
Atribut	
Id : integer	Identifikasi sebuah vertex.
_left : Vertex<T>	Vertex tetangga di sebelah kiri
_up : Vertex<T>	Vertex tetangga di sebelah atas
_right : Vertex<T>	Vertex tetangga di sebelah kanan
_down : Vertex<T>	Vertex tetangga di sebelah bawah
Method	
ConnectLeft : void	Menghubungkan sisi kiri
ConnectUp : void	Menghubungkan sisi atas
ConnectRight : void	Menghubungkan sisi kanan
ConnectDown : void	Menghubungkan sisi bawah

4.3. Implementasi Program

Pseudocode dari Algoritma penyelesaian.

BSF

```
function BFS (input: Graf : Graph, initialState : State, goal : Coordinate,
directionPriority : string, output: List of Coordinates, List of State)

KAMUS LOKAL
states : List of State
State : State
q : Queue of Vertex
v : Vertex
track : Queue of List of Coordinate
path : List of Coordinate
blocked : boolean

ALGORITMA
{ Traversal vertex yang akan dilalui }
while(length(q) > 0)
    { dequeue vertex q dan inisialisasi sebagai v }
    v <- dequeue(q)

    { dequeue Coordinate track dan inisialisasi sebagai t }
    t <- dequeue(track)

    { dapatkan info yaitu nilai dari node v }
    currentLocation(state) <- info(v)

    { cek apakah masih ada kemungkinan arah yang dituju }
    if (length(t) > 1) then
        dir(state) <- DetermineDirection(t[length(t)-2], t[length(t)-1])

    { cek apakah pada node tersebut merupakan goal lalu simpan state
    tersebut dan tambahkan koordinat ke path }
    if (info(v) = goal) then
        states.Add(State(state))
        path <- t
        break

    { cek apakah node tersebut sudah dilewati }
    if (state.IsVisited(info(v))) then
        continue

    { traversal semua arah dengan urutan Right Down Left Up}
    blocked <- true
    dir traversal directionPriority
        { cari tetangga-tetangga dengan implementasi BFS dan memanipulasi
        track dan q lalu mengubah blocked menjadi false }
        Seek(v, q, t, track, state, dir, &blocked)

    { jika sudah tidak memungkinkan untuk lanjut pada jalur itu maka simpan
    state tersebut dan hapus state yang dianggap telah dilewati }
    if (blocked and length(q) = 0) then
```

```

        SaveAllVisitedLocations(state)
        ClearVisitedLocations(state)
        enqueue(q, v)
    else { tambahkan state yang sudah dilewati }
        states.Add(State(state))
        AddVisitedLocation(state, info(v))

Dir(states[0]) <- Dir(states[1])
return (length(path) > 0 ? path : null, states)

```

DFS

```

function DFS (input: Graf : Graph, initialState : State, goal : Coordinate,
directionPriority : string, output: List of Coordinates, List of State)

KAMUS LOKAL
states : List of State
State : State
q : Stack of Vertex
v : Vertex
track : Stack of List of Coordinate
path : List of Coordinate
blocked : boolean

ALGORITMA
{ Traversal vertex yang akan dilalui }
while(length(q) > 0)
    { dequeue vertex q dan inisialisasi sebagai v }
    v <- pop(q)
    { dequeue Coordinate track dan inisialisasi sebagai t }
    t <- pop(track)
    { dapatkan info yaitu nilai dari node v }
    currentLocation(state) <- info(v)

    if (length(t) > 1) then
        dir(state) <- DetermineDirection(t[length(t)-2], t[length(t)-1])

    if (info(v) = goal) then
        states.Add(State(state))
        path <- t
        break

    if (state.IsVisited(info(v)) then
        continue

    dir traversal directionPriority.reversed()
        Seek(v, q, t, track, state, dir, &blocked)

    if (blocked and length(q) = 0) then
        SaveAllVisitedLocations(state)
        ClearVisitedLocations(state)
        enqueue(q, v)
    else
        states.Add(State(state))
        AddVisitedLocation(state, info(v))

```

```

Dir(states[0]) <- Dir(states[1])
return (length(path) > 0 ? path : null, states)

```

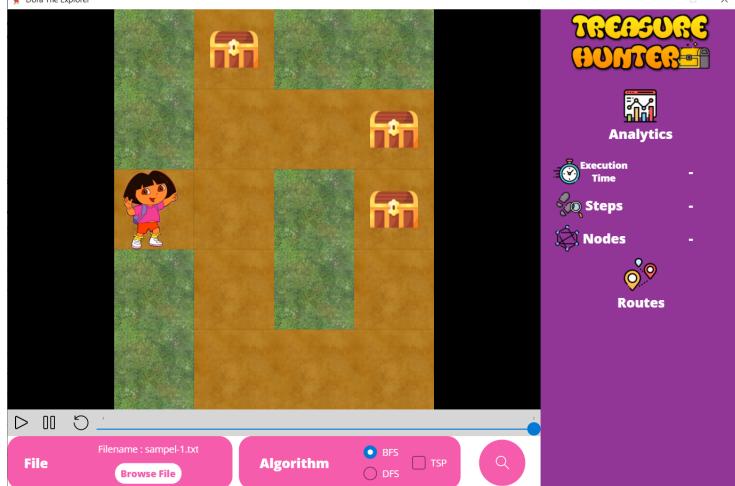
4.4. Eksperimen

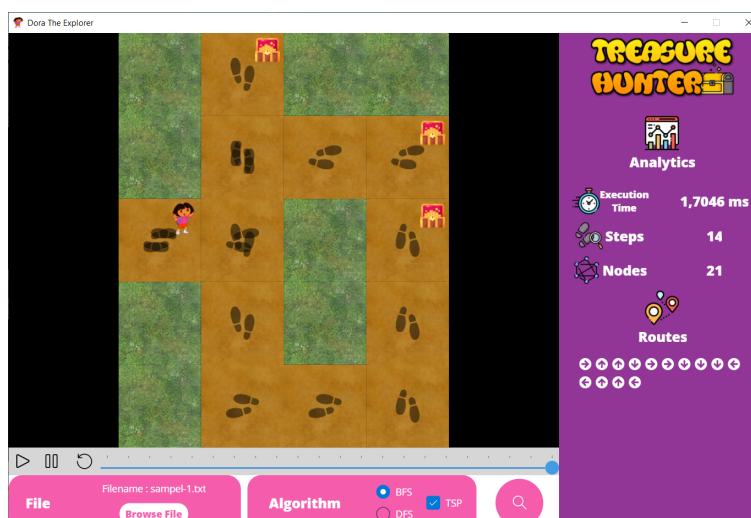
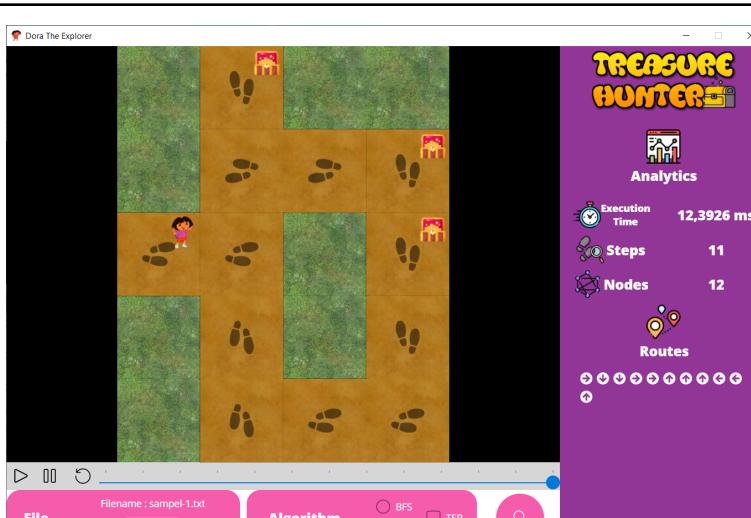
Berikut adalah eksperimen yang dilakukan untuk mengetahui batasan dan kapabilitas dari program DoraTheExplorer. Eksperimen dilakukan dengan file .txt dari *Test Case* yang disediakan oleh spesifikasi tugas. Program dijalankan pada komputer dengan spesifikasi sebagai berikut.

Laptop : HP Pavilion Gaming 15

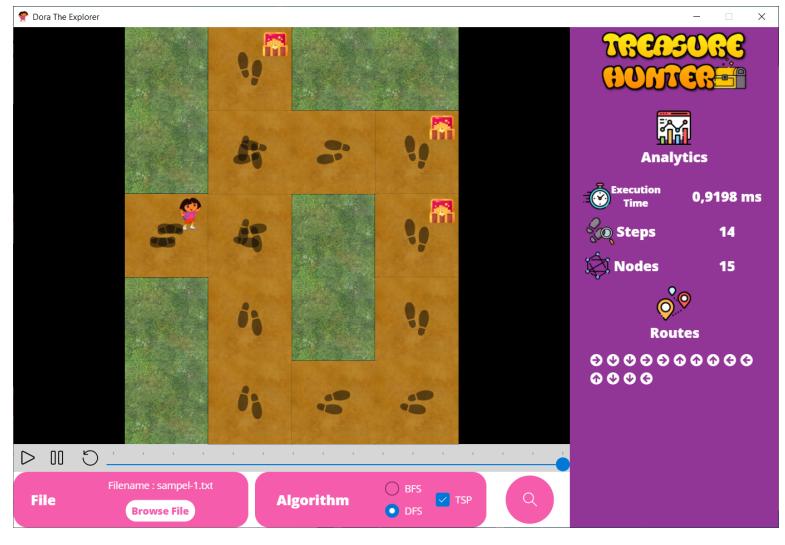
Processor : Intel(R) Core(TM) i5-10300H CPU @ 2.50 GHz

RAM : 16 GB @ 2933 MHz

Sampel 1	
Input File	X T X X X R R T K R X T X R X R X R R R
Visualisasi	 <p>The screenshot shows a grid-based treasure hunt map with various rooms and treasures. A character named Dora is standing in one of the rooms. To the right of the map is a purple sidebar titled "TREASURE HUNTER" with sections for "Analytics" (Execution Time, Steps, Nodes, Routes), and a bottom navigation bar with "File" (Filename: sampel-1.txt, Browse File), "Algorithm" (BFS selected, DFS, TSP), and a search icon.</p>

BFS	
TSP dengan implementasi BFS	
DFS	

TSP dengan implementasi DFS

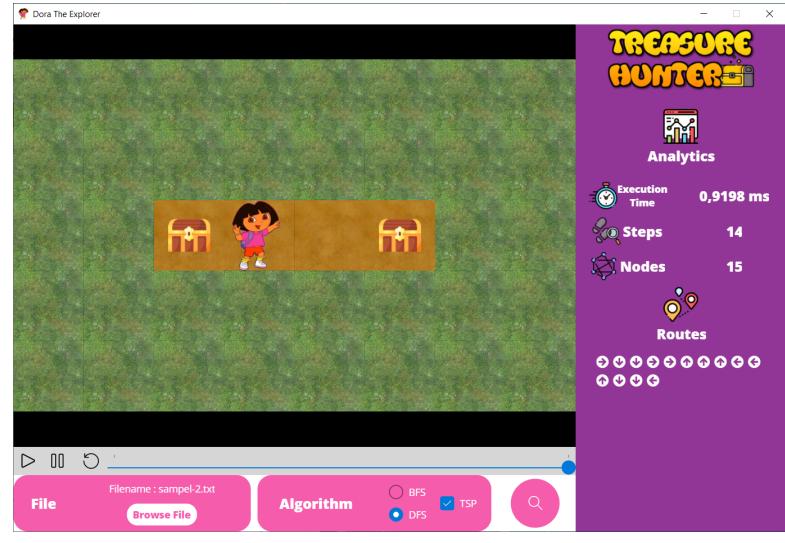


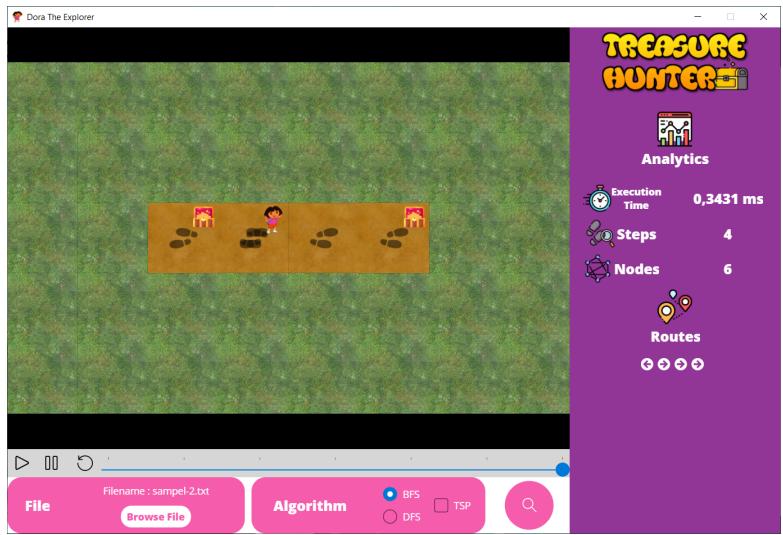
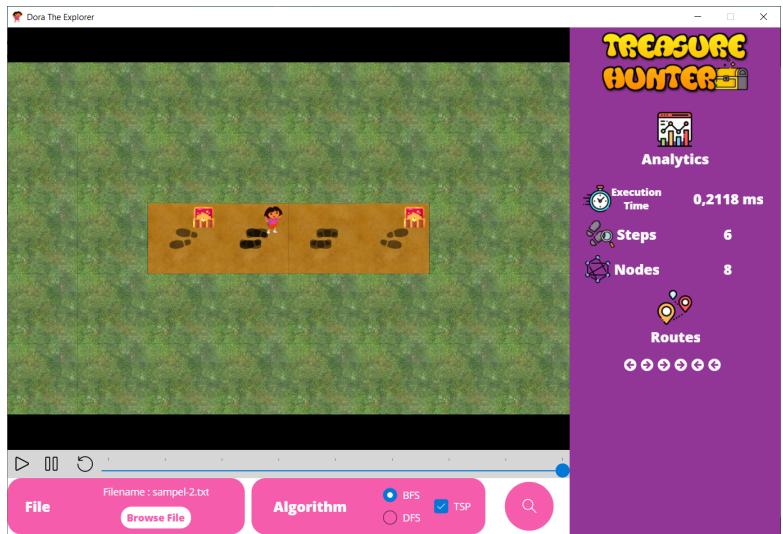
Sampel 2

Input File

```
X X 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 X X
```

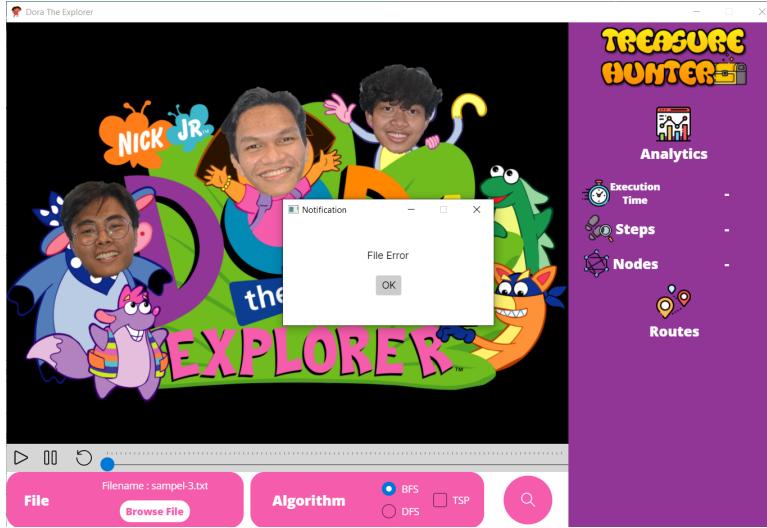
Visualisasi



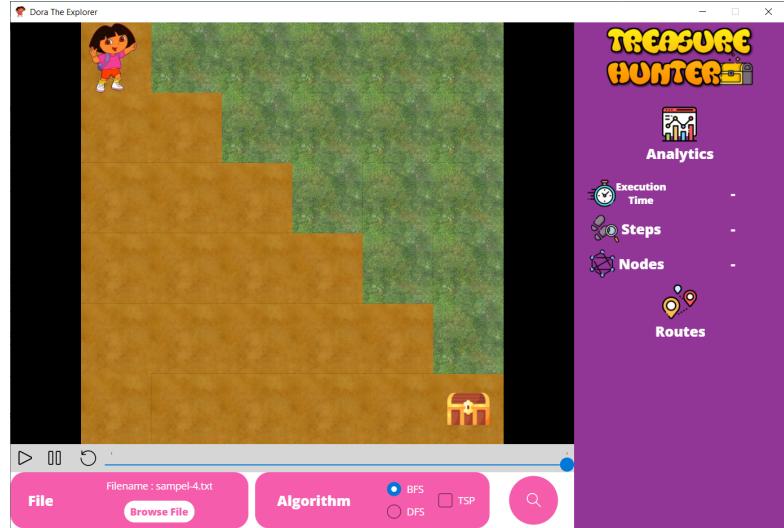
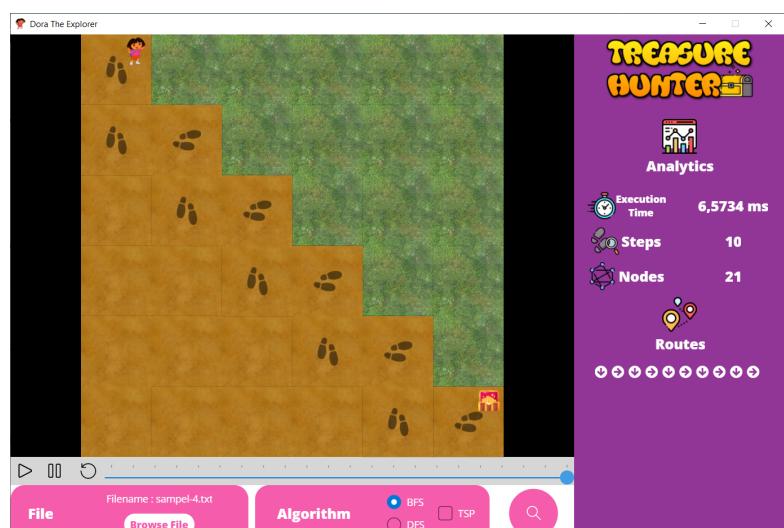
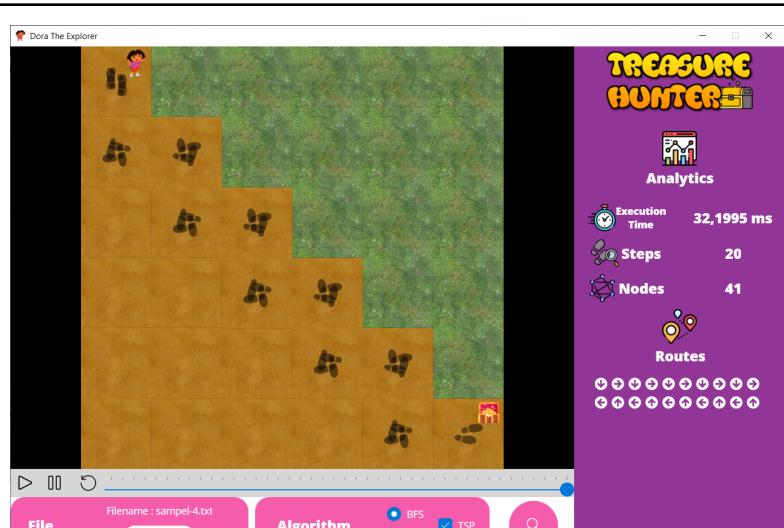
BFS	 <p>Dora The Explorer</p> <p>TREASURE HUNTER</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time: 0,3431 ms Steps: 4 Nodes: 6 Routes:  <p>File Filename : sampel-2.txt Browse File</p> <p>Algorithm <input checked="" type="radio"/> BFS <input type="radio"/> DFS <input type="checkbox"/> TSP</p> <p>Search</p>
TSP dengan implementasi BFS	 <p>Dora The Explorer</p> <p>TREASURE HUNTER</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time: 0,2118 ms Steps: 6 Nodes: 8 Routes:  <p>File Filename : sampel-2.txt Browse File</p> <p>Algorithm <input checked="" type="radio"/> BFS <input type="radio"/> DFS <input checked="" type="checkbox"/> TSP</p> <p>Search</p>
DFS	



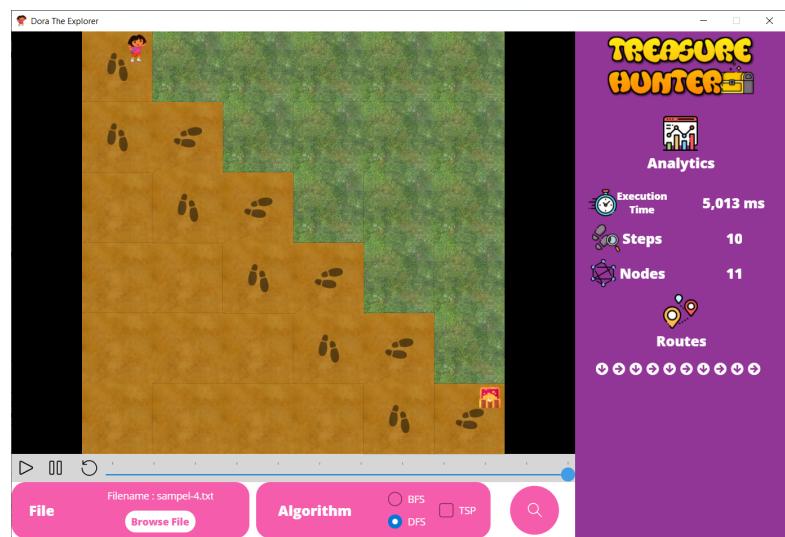
Sampel 3	
Input File	J A N G A N L U P A C E K Y A N G B E G I N I Y

Visualisasi	
BFS	-
TSP dengan implementasi BFS	-
DFS	-
TSP dengan implementasi DFS	-

Sampel 4	
Input File	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

Visualisasi	
BFS	
TSP dengan implementasi BFS	

DFS



TSP dengan
implementasi DFS



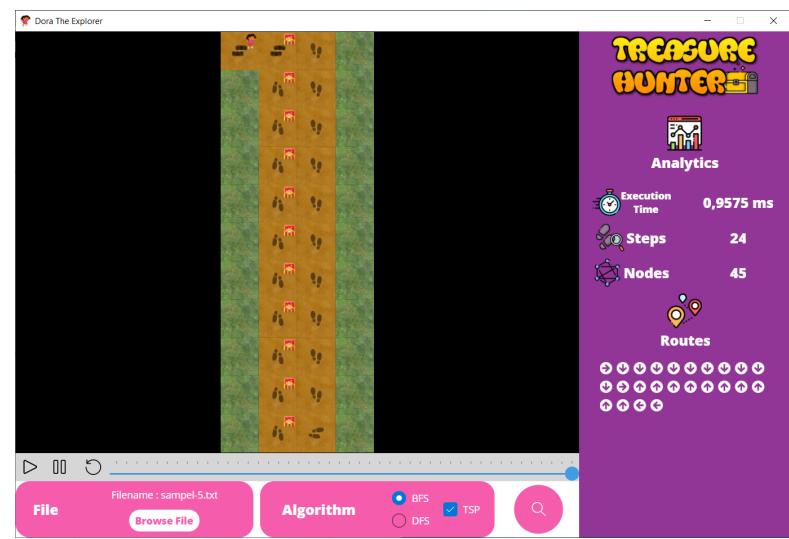
Sampel 5

Input File

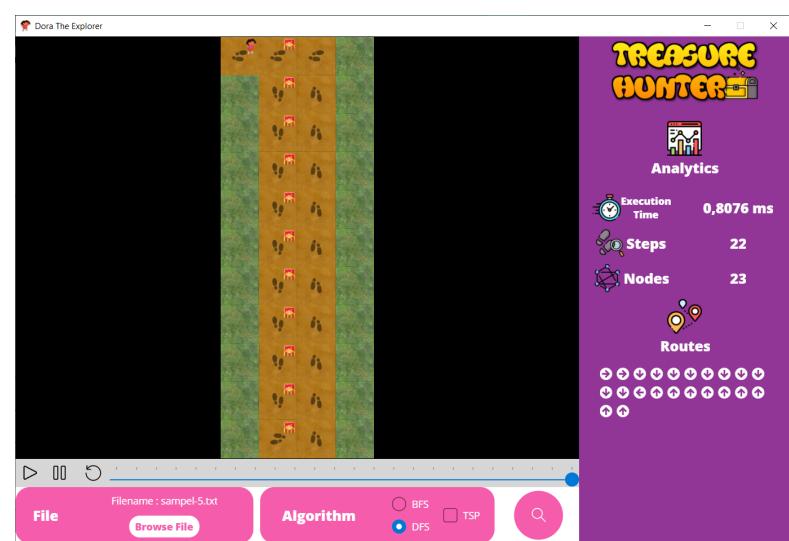
K	T	R	X
X	T	R	X
X	T	R	X
X	T	R	X
X	T	R	X
X	T	R	X
X	T	R	X
X	T	R	X
X	T	R	X
X	T	R	X

	X T R X
Visualisasi	<p>Dora The Explorer</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time: 0.9417 ms Steps: 20 Nodes: 21 Routes: (Sequence of arrows) <p>File Filename : sampel-5.txt Algorithm BFS DFS TSP</p>
BFS	<p>Dora The Explorer</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time: 3,1508 ms Steps: 11 Nodes: 22 Routes: (Sequence of arrows) <p>File Filename : sampel-5.txt Algorithm BFS DFS TSP</p>

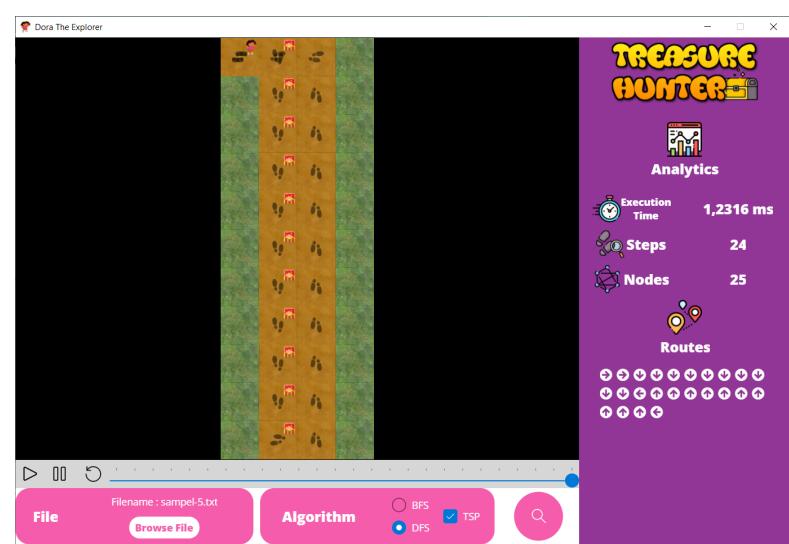
TSP dengan implementasi BFS



DFS



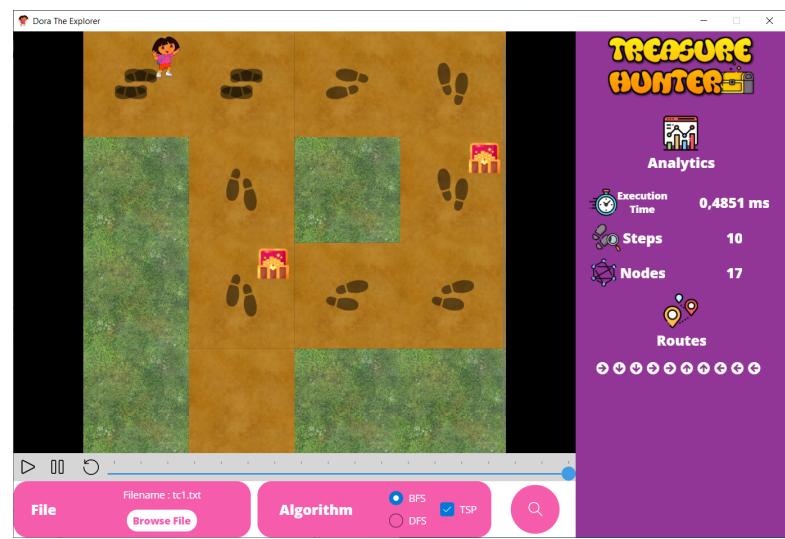
TSP dengan implementasi DFS



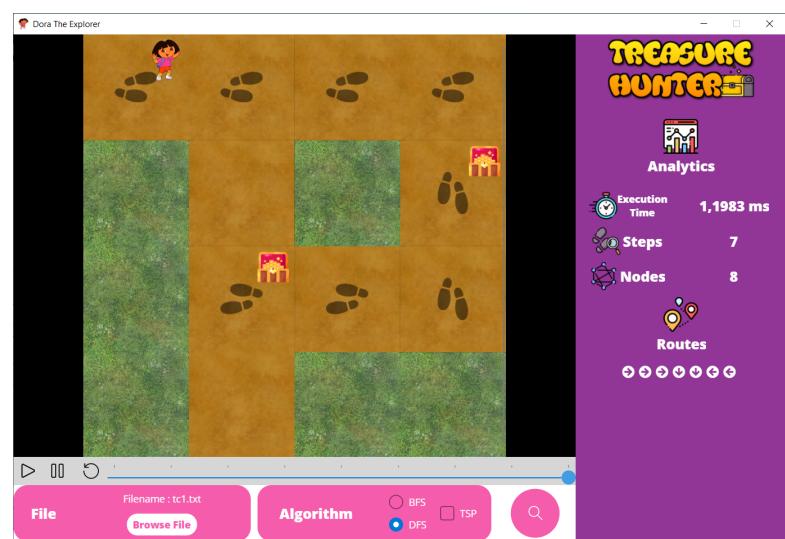
Dibawah ini adalah hasil dari test case yang kelompok kami buat untuk memastikan lebih jauh efektivitas dan efisiensi program DoraTheExplorer.

Test Case 1	
Input File	K R R R X R X T X T R R X R X X
Visualisasi	
BFS	

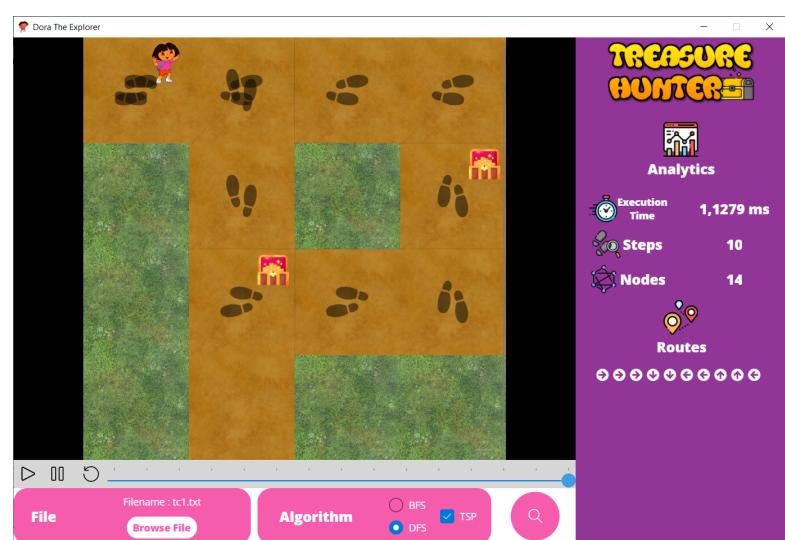
TSP dengan implementasi BFS



DFS



TSP dengan implementasi DFS



Test Case 2	
Input File	<pre> K R R R R X R X T R X T R R R X R R R T X X R R R R R T X X X X R X X X X R R X </pre>
Visualisasi	<p>Dora The Explorer</p> <p>TREASURE HUNTER</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time: 1,1279 ms Steps: 10 Nodes: 14 Routes: <p>File Filename : tc2.txt Algorithm BFS DFS TSP Search</p>
BFS	<p>Dora The Explorer</p> <p>TREASURE HUNTER</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time: 1,184 ms Steps: 13 Nodes: 30 Routes: <p>File Filename : tc2.txt Algorithm BFS DFS TSP Search</p>

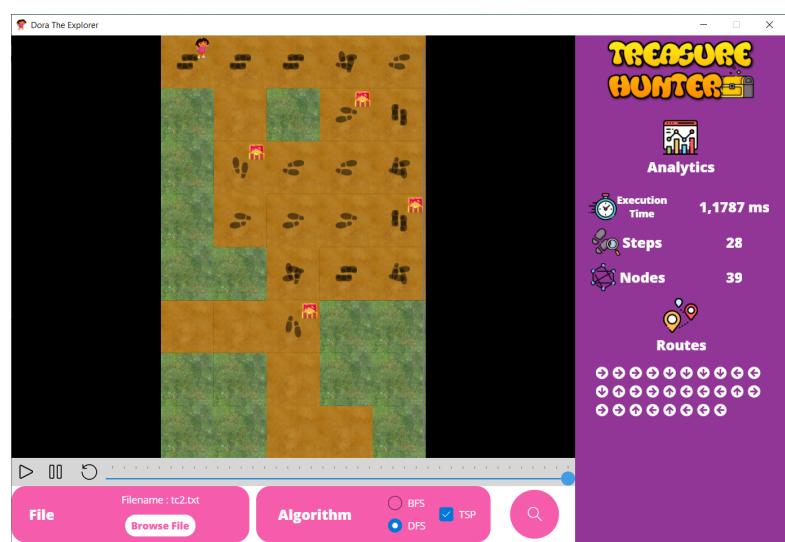
TSP dengan implementasi BFS



DFS

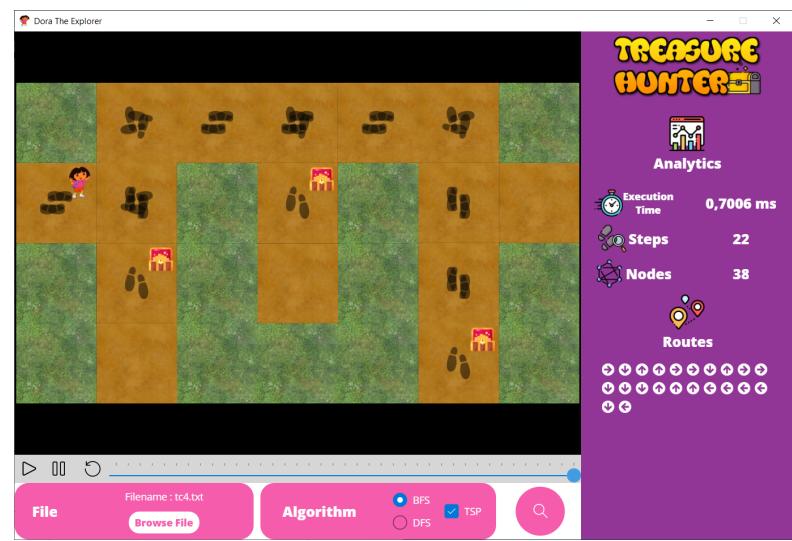


TSP dengan implementasi DFS

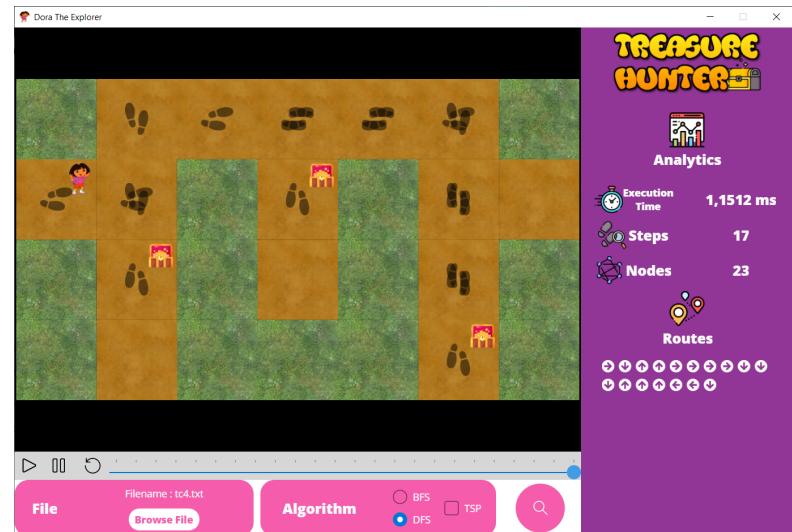


Test Case 4	
Input File	<pre> X R R R R R X K R X T X R R X T X R X R X X R X X X T X </pre>
Visualisasi	<p>Dora The Explorer</p> <p>File Filename : tc4.txt Browse File</p> <p>Algorithm BFS DFS TSP</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time: 1,178 ms Steps: 28 Nodes: 39 Routes: 13
BFS	<p>Dora The Explorer</p> <p>File Filename : tc4.txt Browse File</p> <p>Algorithm BFS DFS TSP</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time: 0,1809 ms Steps: 13 Nodes: 26 Routes: 13

TSP dengan implementasi BFS



DFS



TSP dengan implementasi DFS



Test Case 5	
Input File	<pre> K R R R R R R R R R R X R R R R R R R R T </pre>
Visualisasi	<p>Dora The Explorer</p> <p>TREASURE HUNTER</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time 1.8257 ms Steps 22 Nodes 30 Routes <p>File Filename : tc5.txt Algorithm BFS DFS TSP</p>
BFS	<p>Dora The Explorer</p> <p>TREASURE HUNTER</p> <p>Analytics</p> <ul style="list-style-type: none"> Execution Time 0.1928 ms Steps 18 Nodes 60 Routes <p>File Filename : tc5.txt Algorithm BFS DFS TSP</p>

TSP dengan implementasi BFS



DFS



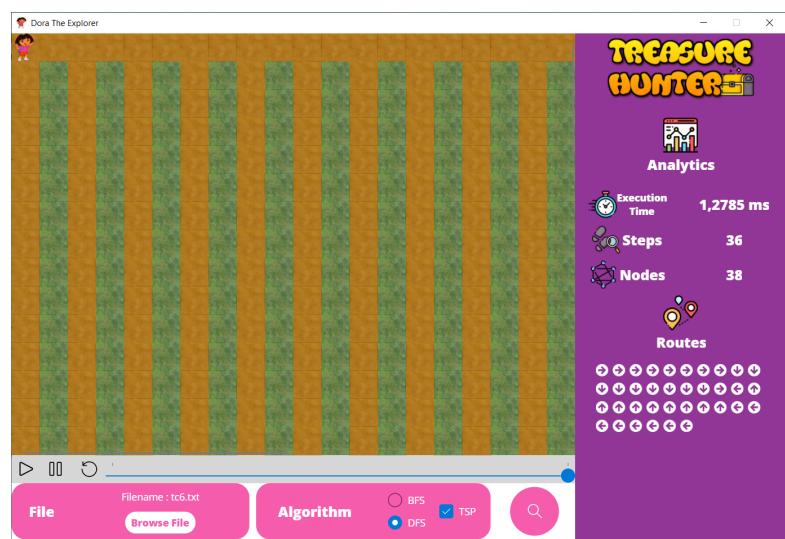
TSP dengan implementasi DFS



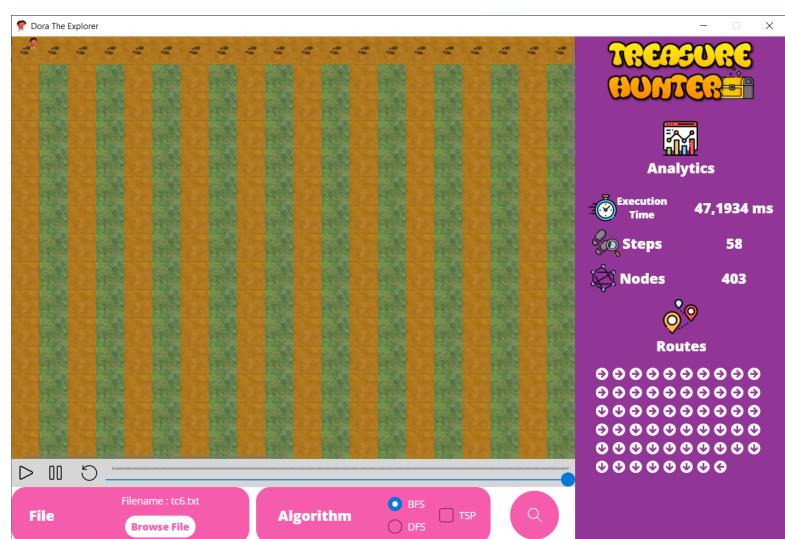
Test Case 6

Input File

Visualisasi



BFS



TSP dengan
implementasi BFS



DFS	 <p>Execution Time: 1,6067 ms Steps: 74 Nodes: 187 Routes:</p>
TSP dengan implementasi DFS	 <p>Execution Time: 19,2159 ms Steps: 130 Nodes: 1011 Routes:</p>

4.5. Analisis Hasil Eksperimen

Dapat dilihat dari hasil test case bahwa program dapat menangani sebagian besar kasus yang dimasukan. Pada sampel 3 untuk kasus file dengan format yang salah, program mengeluarkan pesan error dengan window baru dan komputasi tidak bisa dilanjutkan dengan file yang sama. Untuk kasus normal sisanya, program dapat dengan efektif mencari jalur untuk mendapatkan semua treasure. Namun perbedaan terletak pada algoritmanya. Seringkali pada kasus yang sama algoritma yang berbeda menghasilkan jalur yang berbeda pula. TSP secara umum menghasilkan jumlah step yang lebih banyak dibandingkan algoritma yang tidak menerapkan TSP.

BAB V :Kesimpulan dan Saran

5.1. Kesimpulan

Dengan program hasil penggerjaan tugas besar ini, dapat disimpulkan bahwa algoritma DFS dan BFS dapat secara efektif menyelesaikan permasalahan *Maze Treasure Hunt*. Keduanya memiliki penyelesaian masalah yang unik dan mungkin di suatu kasus DFS lebih optimal dan mungkin pula di kasus lain BFS lebih optimal.

5.2. Saran

Saran kami untuk pelaksanaan tugas besar 2 ini adalah untuk tidak memberikan waktu penggerjaan tugas bersamaan dengan pelaksanaan ujian tengah semester. Mengingat bahwa UTS merupakan salah satu komponen yang penting dalam penilaian akademik, beban tugas besar yang seharusnya dikerjakan dalam 3 minggu seolah-olah hanya menjadi 2 minggu karena para mahasiswa diharuskan untuk fokus menghadapi UTS. Jika sekiranya diharuskan melaksanakan tugas besar pada saat UTS maka langkah lebih baik jika beban dan kompleksitas tugas diturunkan.

5.3. Refleksi

1. Terlalu berfokus kepada tugas besar II strategi algoritma yang membuat tugas besar lain terlantarkan.
2. Mempelajari *framework* dan bahasa pemrograman baru dapat menambahkan wawasan tentang informatika.
3. Membuat perasaan puas dengan hasil algoritma dan *interface* pada tugas besar kali ini yang telah penulis buat.

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>

<https://docs.avaloniaui.net/>

https://en.wikipedia.org/wiki/Graph_traversal

https://en.wikipedia.org/wiki/Depth-first_search

https://en.wikipedia.org/wiki/Breadth-first_search

Lampiran

Link Repository GitHub : https://github.com/razzanYoni/Tubes2_DoraTheExplorer

Link Video : <https://youtu.be/c3-eqOBqVXg>