

LAPORAN TUGAS BESAR 2

Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt



**Disusun Oleh:
Kelompok 49**

**Anggota Kelompok:
Ezra Maringan Christian Mastra Hutagaol - 13521073
Ammar Rasyad Chaeroel - 13521136
Zidane Firzatullah - 13521163**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
MARET 2023**

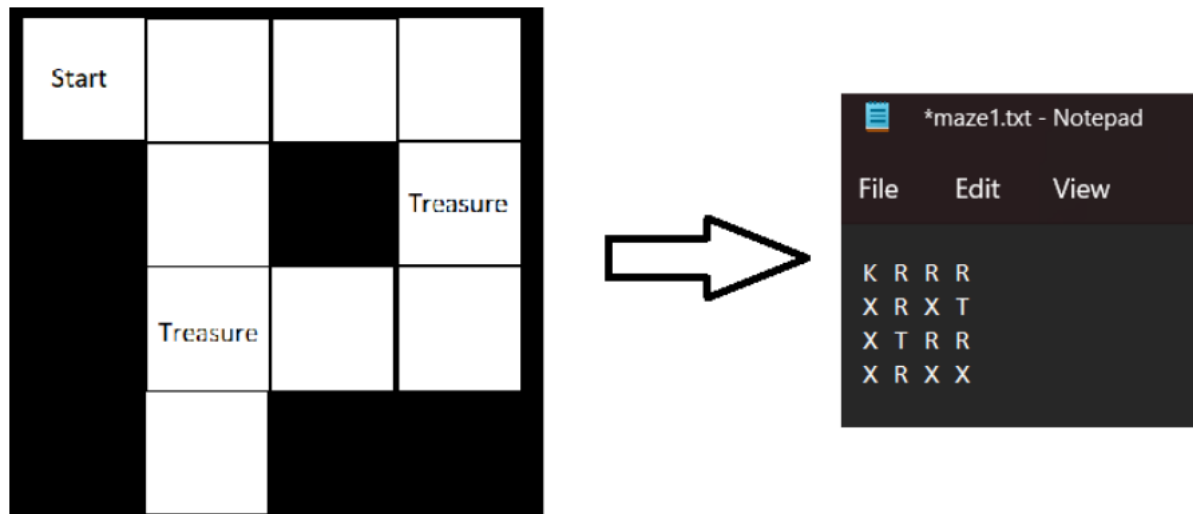
Bab 1

Deskripsi Tugas

Dalam tugas besar ini, Anda akan 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 :

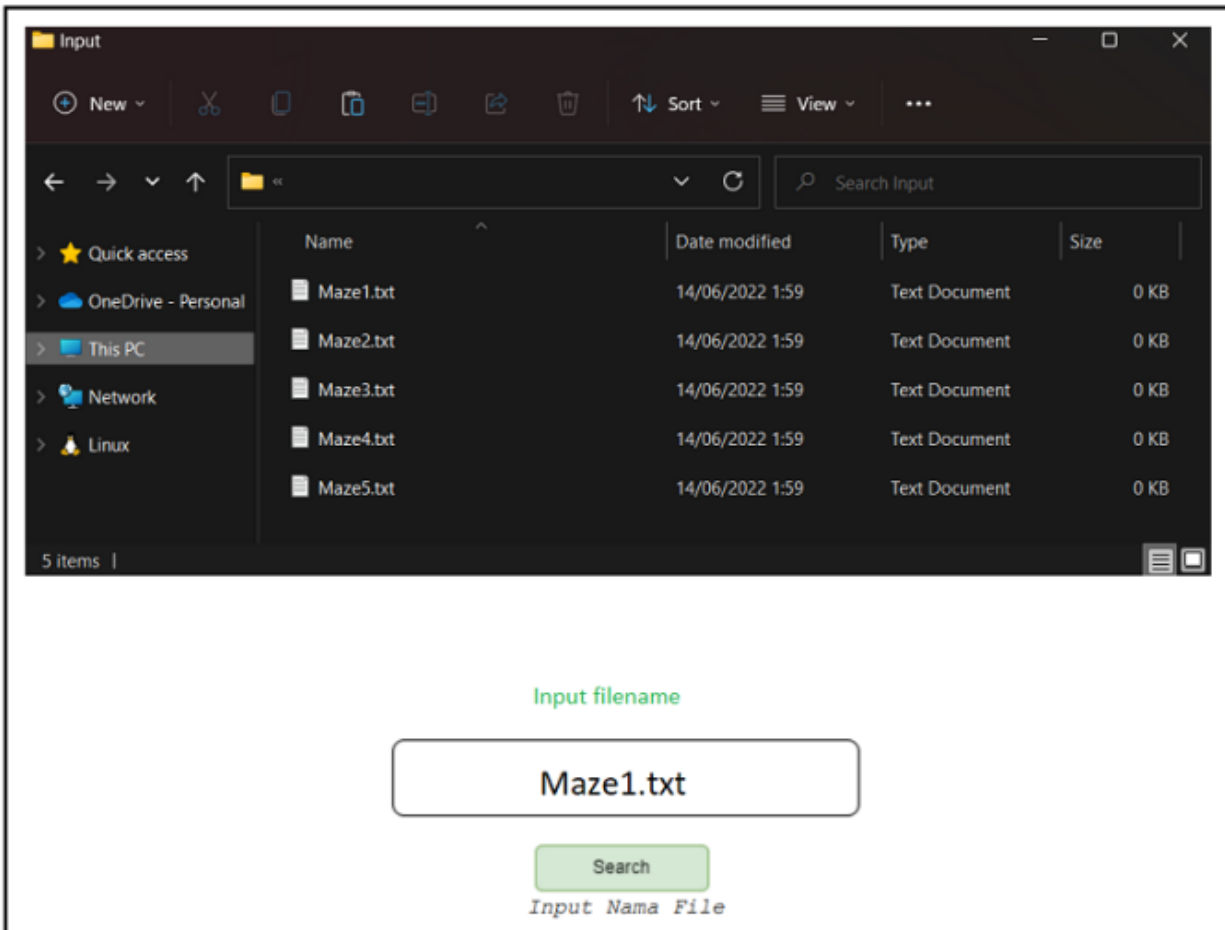


Gambar 1.1 Ilustrasi input file maze

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.

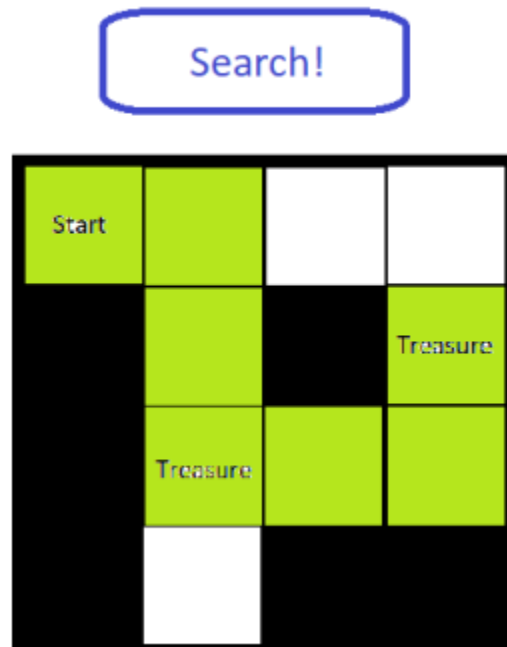
Contoh input aplikasi :



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 1

Setelah program melakukan pembacaan input, program akan memvisualisasikan gridnya 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

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

Treasure Hunt Solver

Input

Filename

Algoritma
☒ BFS
☐ DFS

Output

Start			
			Treasure
	Treasure		

Route:
Steps:

Nodes :
Execution Time :

Gambar 1.4 Tampilan program sebelum dicari solusinya

Treasure Hunt Solver

Input

Filename

Algoritma
☒ BFS
☐ DFS

Output

Start			
			Treasure
	Treasure		

Route: R - D - D - R - R - U
Steps: 6

Nodes : 11
Execution Time : 850 ms

Gambar 1.5 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.

Program yang dibuat harus memenuhi spesifikasi wajib sebagai berikut:

1. Buatlah program dalam bahasa C# untuk mengimplementasi Treasure Hunt Solver sehingga diperoleh output yang diinginkan. Penelusuran harus memanfaatkan algoritma BFS dan DFS.
2. Awalnya program menerima file atau nama file maze treasure hunt.
3. 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.
4. Pengguna memilih algoritma yang digunakan menggunakan toggle yang tersedia
5. Program kemudian dapat menampilkan visualisasi akhir dari maze (dengan pewarnaan rute solusi).
6. Program menampilkan luaran berupa durasi eksekusi, rute solusi, banyaknya langkah, serta banyaknya node yang diperiksa.

7. Mahasiswa tidak diperkenankan untuk melihat atau menyalin library lain yang mungkin tersedia bebas terkait dengan pemanfaatan BFS dan DFS. Akan tetapi, untuk algoritma lain diperbolehkan menggunakan library jika ada.

Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk memvisualisasikan matrix dalam bentuk grid adalah DataGridView. Berikut adalah panduan singkat terkait penggunaannya <http://csharp.net-informations.com/datagridview/csharp-datagridview-tutorial.htm>

Bab 2

Landasan Teori

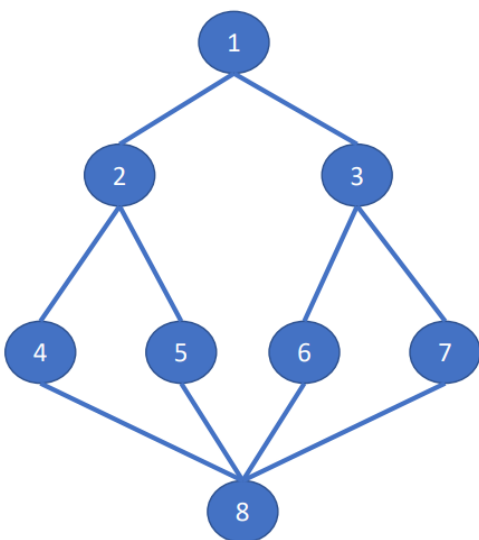
Bab 2.1 Dasar Teori

Algoritma traversal graf adalah algoritma yang mengunjungi simpul dengan cara sistematis. Algoritma traversal graf terbagi menjadi 2 yaitu pencarian melebar (breadth first search / BFS) dan pencarian mendalam (depth first search). Pada algoritma ini kita mengasumsikan graf terhubung

Breadth-First search atau BFS merupakan salah satu algoritma yang paling sederhana dalam melakukan pencarian pada graf dan pola dasar pada algoritma pencarian graf lainnya. Algoritma *minimum-spanning-tree* Prim dan Algoritma pencarian jarak terdekat Dijkstra menggunakan ide yang serupa dengan pencarian *breadth-first search*. Dalam algoritma *breadth-first search*, graf direpresentasikan dalam bentuk $G = (V, E)$ dengan v merupakan simpul awal penelusuran

Berikut merupakan algoritma BFS :

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



Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

Urutan simpul yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

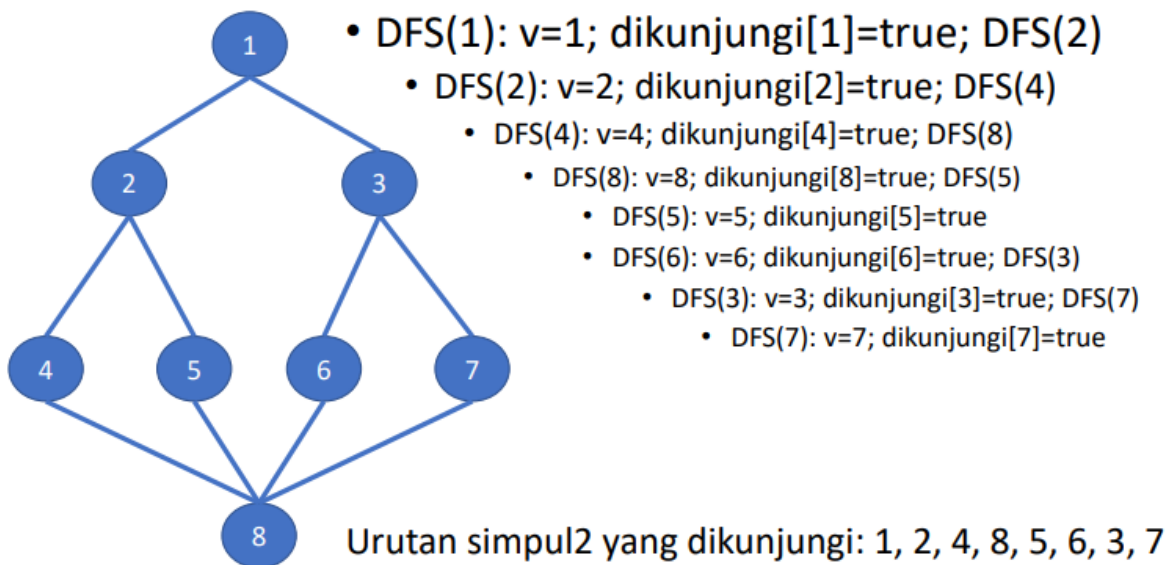
Gambar 2.1 Ilustrasi BFS

Berbeda dengan *breadth-first search*, algoritma pencarian *depth-first search* melakukan penelusuran terlebih dahulu ke-"dalam" ketika memungkinkan. Algoritma *depth-first search*

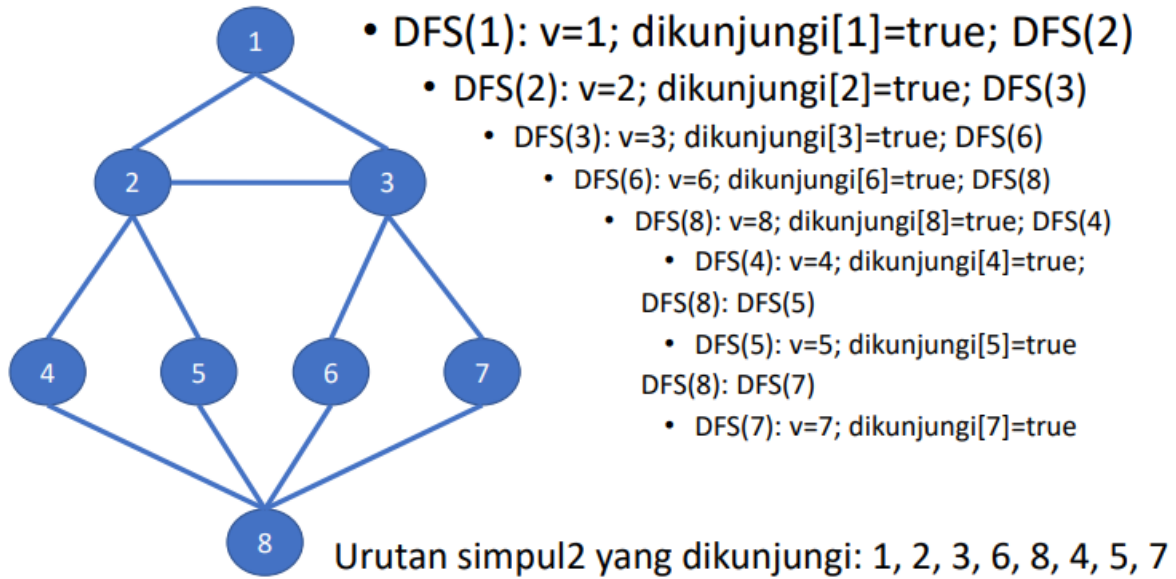
akan menelusuri simpul tetangga dari simpul yang ditelusuri sekarang. Setelah sudah tidak ada simpul tetangga yang tersedia, algoritma akan melakukan *backtracks* untuk menelusuri simpul-simpul tetangga dari simpul sebelumnya. Pada DFS v merupakan simpul awal penelusuran juga.

Berikut merupakan algoritma DFS :

1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v
3. Ulangi DFS mulai dari simpul w
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi



Gambar 2.2 Ilustrasi DFS



Gambar 2.3 Ilustrasi DFS

2.2 C# Desktop Application Development

C# adalah bahasa pemrograman yang dibuat oleh Microsoft dan ditargetkan berjalan di atas platform .NET (dotnet). Program C# tidak seperti program C dan C++ yang dicompile menjadi bahasa assembly dan bisa dieksekusi langsung oleh prosesor. Program C# dicompile menjadi CIL (Common Intermediate Language). Bahasa C# menggunakan paradigma pemrograman berorientasi objek dan banyak digunakan dalam pembuatan aplikasi berbasis desktop.

Pengembangan *desktop application*, terlebih dalam hal *user interface* (UI), dipermudah dengan menggunakan WinForms/WPF dan bantuan *integrated development environment* (IDE) Visual Studio. WPF digunakan untuk pengembangan *desktop application* berbasis Windows dengan menggunakan bahasa nativenya, yaitu bahasa pemrograman C#

Bab 3

Landasan Teori

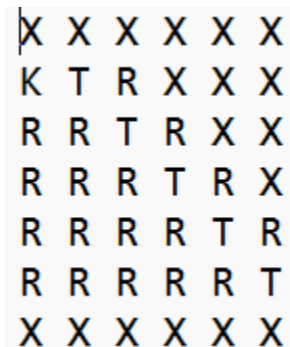
3.1 Langkah-Langkah Pemecahan Masalah

Di dalam Maze terdapat K (Titik awal) ditandai dengan warna biru, T (Treasure) ditandai dengan warna kuning, R (Grid yang mungkin diakses) ditandai dengan warna putih, dan X (Grid halangan yang tidak dapat diakses) ditandai dengan warna hitam. Dari maze yang diberikan dari input akan dilakukan pencarian rute untuk menuju ke T (treasure). Untuk Grid yang sedang dilalui akan ditandai dengan warna kuning. Kemudian jika T (Treasure) sudah ditemukan akan ditandai dengan warna hijau. Untuk mencari rute, pengguna bisa memilih algoritma DFS atau algoritma BFS dalam pencarian.

3.2 Mapping persoalan

Maze dapat digambarkan sebagai sebuah pohon. *Maze* yang terbentuk akan bergantung pada input pengguna. Pada *Maze* yang terbentuk *grid* yang akan ditelusuri bergantung pada algoritma yang dipilih BFS atau DFS.

3.3 Ilustrasi Kasus Lain



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

Gambar 3.3.1 Test case lain

Treasure Hunt Solver

×

Input

Filename

Browse

Algoritma

☒ BFS
☐ DFS
☐ TSP

Durasi Tick

50ms

Output

Start	Treasure				
		Treasure			
			Treasure		
				Treasure	
					Treasure

Search

Route: R-R-D-R-D-R-D-R-D
Steps: 10
Nodes: 31
Time: 0 ms

Gambar 3.3.2 Pengujian Test Case Lain dengan BFS

Treasure Hunt Solver

×

Input

Filename

Browse

Algoritma

☐ BFS
☒ DFS
☐ TSP

Durasi Tick

50ms

Output

Start	Treasure				
		Treasure			
			Treasure		
				Treasure	
					Treasure

Search

Route: R-L-D-R-R-L-L-D-R-R-R-L-L-L-D-R-R-
R-R-L-L-L-L-D-R-R-R-R-R
Steps: 30
Nodes: 56
Time: 0 ms

Gambar 3.3.2 Pengujian Test Case Lain dengan DFS

Bab 4

Analisis Pemecahan Masalah

4.1 Implementasi Program

4.1.1 Kelas BreadthSolver

```
public class BreadthSolver : Solver<Tuple<bool, int, int>>
{
    // unfortunately tuple is immutable, so we have to create a new tuple to sign the sequence

    private static List<Tuple<bool, int, int>> TraceSequence(Tuple<bool, int, int>[,] dynMap,
                                                            in List<Tuple<bool, int, int>> sequence, int startIdx)
    {
        int x ← sequence[startIdx-1].Item2, y ← sequence[startIdx-1].Item3;
        var sequenceN = new List<Tuple<bool, int, int>>();

        for (var i = startIdx - 1; i >= 0; --i)
        {
            int currentX = sequence[i].Item2, currentY = sequence[i].Item3;

            if (currentX != x || currentY != y) continue;
            sequenceN.Insert(0, sequence[i]);
            var tuple ← dynMap[x, y];
            x ← tuple.Item2; y ← tuple.Item3;
        }

        return sequenceN;
    }

    protected override Tuple<int, int> MergerItemGetter(Tuple<bool, int, int> item)
    {
        return Tuple.Create(item.Item2, item.Item3);
    }

    protected override bool IsFound(int idx1, int idx2, Tuple<bool, int, int> item)
    {
        {
            var (_, x, y) ← item;
            if (idx1 == x && idx2 == y)
            {
                return true;
            }
        }
        return false;
    }

    public override Solution Solve(in bool tsp)
    {
        var map ← TreasureMap.MapArr;
```

```

var (startIdx1, startIdx2) ← TreasureMap.StartPoint;
var treasureCount ← TreasureMap.TreasureCount;
var size ← map.GetLength(0);

var trace ← new Tuple<bool, int, int>[map.GetLength(0), map.GetLength(1)];
CreateOrClearTrace(trace, map, Tuple.Create(true, 0, 0), Tuple.Create(false, 0, 0));

var sequence ← new List<Tuple<bool, int, int>>();
var finalSequence ← new List<Tuple<int, int>>();
var treasureSet ← new HashSet<int>();

sequence.Add(Tuple.Create(false, startIdx1, startIdx2));
trace[0, 0] ← Tuple.Create(true, trace[0,0].Item2, trace[0,0].Item3);
int treasureFound ← 0, leftIdx ← 0, rightIdx ← 1, nodesCheckedCount ← 0;

var startTime ← DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond;

while (leftIdx < rightIdx)
{
    ++nodesCheckedCount;
    var (_, idx1, idx2) ← sequence[leftIdx++];

    if (map[idx1, idx2] == 'T' && !treasureSet.Contains(idx1 * size + idx2))
    {
        treasureSet.Add(idx1 * size + idx2);
        ++treasureFound;

        var pSequence ← TraceSequence(trace, sequence, leftIdx);
        sequence ← pSequence;
        leftIdx ← rightIdx ← pSequence.Count;

        CreateOrClearTrace(trace, map, Tuple.Create(true, 0, 0), Tuple.Create(false, 0, 0));
        trace[idx1, idx2] ← Tuple.Create(true, trace[idx1, idx2].Item2, trace[idx1, idx2].Item3);

        MergeSequenceWithDuplicateEnds(finalSequence, sequence);
    }

    if (treasureFound == treasureCount)
    {
        if (tsp && map[startIdx1, startIdx2] != 'T')
        {
            --treasureFound;
            map[startIdx1, startIdx2] = 'T';
        }
        else break;
    }

    var tuple ← Tuple.Create(true, idx1, idx2);

    if (idx2 - 1 >= 0 && trace[idx1, idx2 - 1].Item1 == false)

```

```

    {
        trace[idx1, idx2 - 1] = tuple;
        sequence.Add(Tuple.Create(true, idx1, idx2-1)); ++rightIdx;
    }

    if (idx2 + 1 < map.GetLength(1) && trace[idx1, idx2 + 1].Item1 == false)
    {
        trace[idx1, idx2 + 1] ← tuple;
        sequence.Add(Tuple.Create(true, idx1, idx2+1)); ++rightIdx;
    }

    if (idx1 - 1 >= 0 && trace[idx1 - 1, idx2].Item1 == false)
    {
        trace[idx1-1, idx2] ← tuple;
        sequence.Add(Tuple.Create(true, idx1-1, idx2)); ++rightIdx;
    }

    if (idx1 + 1 < map.GetLength(0) && trace[idx1 + 1, idx2].Item1 == false)
    {
        trace[idx1+1, idx2] ← tuple;
        sequence.Add(Tuple.Create(true, idx1+1, idx2)); ++rightIdx;
    }

}

var endTime ← DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond;

if (tsp) map[startIdx1, startIdx2] = 'K';

return new Solution
{
    Path ← TracePath(finalSequence),
    Sequence ← finalSequence,
    ExecutionTime ← endTime-startTime,
    NodesCheckedCount ← nodesCheckedCount
};
}

// example
public static void Main()
{
    var arr ← new char[,] { { 'K', 'R', 'R', 'R' }, { 'X', 'R', 'X', 'T' }, { 'T', 'T', 'R', 'R' }, { 'X', 'R', 'X', 'X' } };

    var treasureMap ← new TreasureMap()
    {
        MapArr ← arr,
        StartPoint ← Tuple.Create(0, 0),
        TreasureCount ← 3
    };
}

```

```

var x ← new BreadthSolver {TreasureMap = treasureMap};

var solution ← x.Solve(true);
solution.Sequence.ForEach(Console.Write);
}

}

```

4.1.2 Kelas DepthSolver

```

public class DepthSolver : Solver<Tuple<int, int>>
{
    private bool dfs(in char[,] map, bool[,] visited,
                    in int idx1, in int idx2, ref int nodesCheckedCount,
                    HashSet<int> treasureSet, List<Tuple<int, int>> sequence)
    {
        ++nodesCheckedCount;
        visited[idx1, idx2] ← true;
        int setIndex ← idx1 * map.GetLength(0) + idx2;

        if (map[idx1, idx2] == 'T' && !treasureSet.Contains(setIndex))
        {
            sequence.Insert(0, Tuple.Create(idx1, idx2));
            treasureSet.Add(setIndex);
            return true;
        }

        var res ← false;

        if (idx2 - 1 >= 0 && !visited[idx1, idx2 - 1])
            res ← dfs(map, visited, idx1, idx2 - 1, ref nodesCheckedCount, treasureSet, sequence);

        if (idx2 + 1 < map.GetLength(1) && !visited[idx1, idx2 + 1] && !res)
            res ← dfs(map, visited, idx1, idx2 + 1, ref nodesCheckedCount, treasureSet, sequence);

        if (idx1 - 1 >= 0 && !visited[idx1 - 1, idx2] && !res)
            res ← dfs(map, visited, idx1 - 1, idx2, ref nodesCheckedCount, treasureSet, sequence);

        if (idx1 + 1 < map.GetLength(0) && !visited[idx1 + 1, idx2] && !res)
            res ← dfs(map, visited, idx1 + 1, idx2, ref nodesCheckedCount, treasureSet, sequence);

        if (res)
        {
            var tuple ← Tuple.Create(idx1, idx2);
            sequence.Insert(0, tuple);
        }
    }
}

```



```

    return res;
}

protected override Tuple<int, int> MergerItemGetter(Tuple<int, int> item)
{
    return item;
}

protected override bool IsFound(int idx1, int idx2, Tuple<int, int> item)
{
    var (x, y) ← item;
    if (idx1 == x && idx2 == y) return true;
    return false;
}

public override Solution Solve(in bool tsp)
{
    var map ← TreasureMap.MapArr;
    var (startIdx1, startIdx2) ← TreasureMap.StartPoint;
    var treasureCount ← TreasureMap.TreasureCount;

    List<Tuple<int, int>> sequence = new(), tempSequence = new();
    var visited ← new bool[map.GetLength(0), map.GetLength(1)];
    CreateOrClearTrace(visited, map, true, false);
    int treasureFound ← 0, idx1 ← startIdx1, idx2 ← startIdx2, nodesCheckedCount ← 0;
    var treasureSet ← new HashSet<int>();

    var startTime ← DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond;

    while (treasureCount != treasureFound)
    {
        var avail ← dfs(map, visited, idx1, idx2, ref nodesCheckedCount, treasureSet, tempSequence);

        if (avail)
        {
            ++treasureFound;
            // merge list
            MergeSequenceWithDuplicateEnds(sequence, tempSequence);
            // update starting x and y
            (idx1, idx2) ← tempSequence.Last();
        }

        // clear visited
        CreateOrClearTrace(visited, map, true, false);

        if (tsp && treasureCount == treasureFound)
        {
            if (map[startIdx1, startIdx2] == 'T')
            {

```

```

        break;
    }
    --treasureFound;
    map[startIdx1, startIdx2] = 'T';
}

// clear temp
tempSequence.Clear();
}

var endTime ← DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond;

if (tsp)
{
    map[startIdx1, startIdx2] ← 'K';
}

return new Solution
{
    Path = TracePath(sequence),
    Sequence ← sequence,
    ExecutionTime ← endTime-startTime,
    NodesCheckedCount ← nodesCheckedCount
};
}

public static void Main()
{
    var arr ← new char[,]
    {
        {'K', 'R', 'R', 'R', 'R', 'R', 'X'},
        {'X', 'R', 'X', 'T', 'X', 'R', 'R'},
        {'X', 'T', 'X', 'R', 'X', 'R', 'X'},
        {'X', 'R', 'X', 'X', 'X', 'T', 'X'}
    };

    var treasureMap ← new TreasureMap()
    {
        MapArr ← arr,
        StartPoint ← Tuple.Create(0, 0),
        TreasureCount ← 3
    };

    var x ← new DepthSolver {TreasureMap = treasureMap};

    var solution ← x.Solve(true);
    solution.Sequence.ForEach(Console.Write);
}
}

```

4.1.3 Kelas Solver

```
public abstract class Solver<T>
{
    public TreasureMap TreasureMap;

    protected abstract Tuple<int, int> MergerItemGetter(T item);
    protected abstract bool IsFound(int idx1, int idx2, T item);
    public abstract Solution Solve(in bool tsp);

    protected void MergeSequenceWithDuplicateEnds(List<Tuple<int, int>> sequence, List<T>
addition)
    {
        if (sequence.Count == 0)
        {
            foreach (var tuple in addition)
            {
                sequence.Add(MergerItemGetter(tuple));
            }
            return;
        }

        var (lastX, lastY) ← sequence.Last(); var found ← false;
        sequence.RemoveAt(sequence.Count-1);

        foreach (var tuple in addition)
        {
            found = found ? found : IsFound(lastX, lastY, tuple);
            if (!found) continue;
            sequence.Add(MergerItemGetter(tuple));
        }
    }

    protected static void CreateOrClearTrace<T>(T[,] trace, in char[,] map, T xValue, T nonXValue)
    {
        for (var i = 0; i < trace.GetLength(0); i += 1)
        {
            for (var j = 0; j < trace.GetLength(1); j += 1)
            {
                if (map[i, j] == 'X')
                {
                    trace[i, j] ← xValue;
                }
                else
                {
                    trace[i, j] ← nonXValue;
                }
            }
        }
    }
}
```

```

    }
}

protected static List<char> TracePath(in List<Tuple<int, int>> sequence)
{
    var pathList ← new List<char>();
    for (var i = 0; i < sequence.Count - 1; ++i)
    {
        pathList.Add(GetDirection(sequence[i], sequence[i+1]));
    }
    return pathList;
}

private static char GetDirection(Tuple<int, int> initialPosition, Tuple<int, int> finalPosition)
{
    int inX ← initialPosition.Item1, inY ← initialPosition.Item2,
        finX ← finalPosition.Item1, finY ← finalPosition.Item2;

    if (finX > inX)
    {
        return 'D';
    }

    if (finX < inX)
    {
        return 'U';
    }

    if (finY > inY)
    {
        return 'R';
    }

    if (finY < inY)
    {
        return 'L';
    }

    return 'S';
}
}

```

4.2 Struktur Data

4.2.1 Kelas BreadthSolver

Kelas BreadthSolver merupakan *Child* dari kelas solver dan implementasi dari algoritma BFS. Kelas ini mempunyai method *TraceSequence()*, *MergerItemGetter()*, *IsFound()*, *Main()*.

4.2.2 Kelas DepthSolver

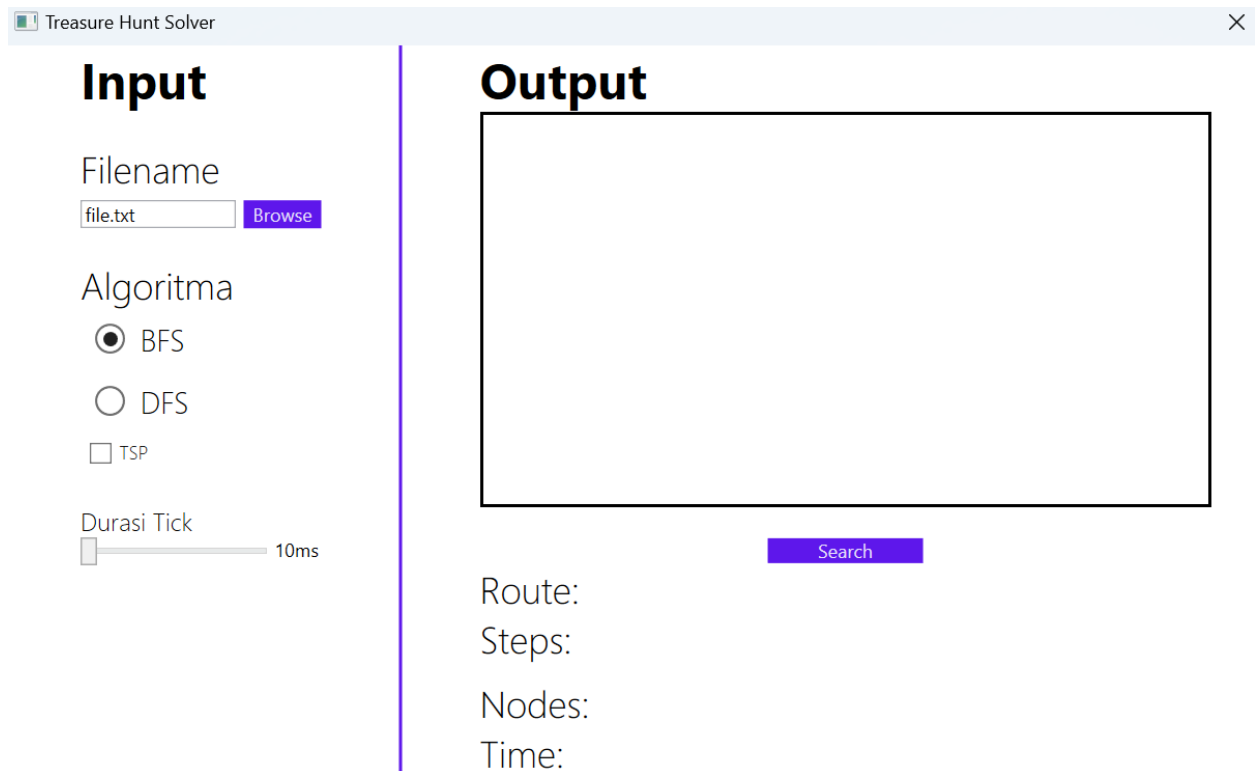
Kelas DepthSolver merupakan *Child* dari kelas solver dan implementasi dari algoritma DFS. Kelas ini mempunyai method *dfs()*, *MergerItemGetter()*, *IsFound()*, *Main()*.

4.2.3 Kelas Solver

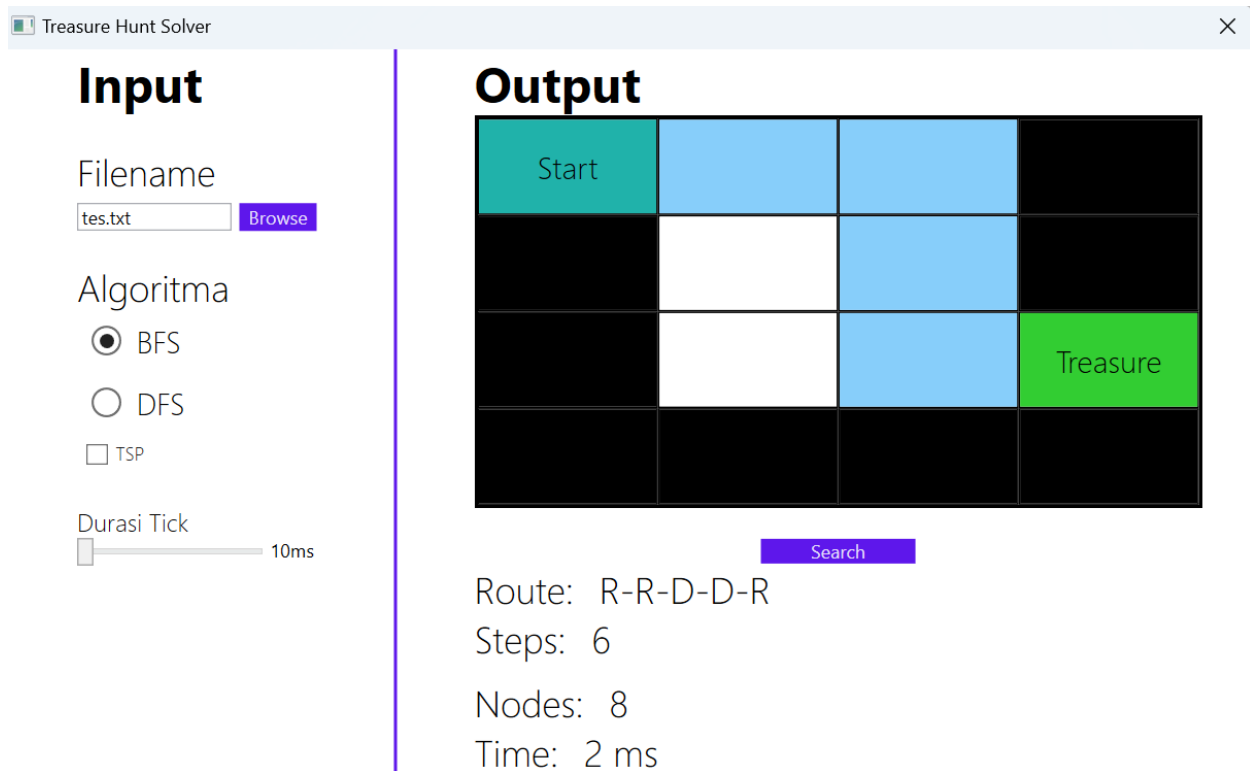
Kelas Solver merupakan *Parent* dari kelas BreadthSolver dan kelas DepthSolver. Kelas ini mempunyai method *MergeSequenceWithDuplicateEnds()*, *CreateOrClearTrace()*, *TracePath()*, *GetDirection()*.

4.3 Tata Cara Penggunaan Program

Untuk menjalankan program, pengguna cukup menjalankan “GUI.exe” yang berada di dalam folder bin. Setelah menjalankan program, pengguna akan diminta untuk melakukan input file txt pada button *Browse*. Setelah melakukan input file txt pengguna bisa memilih ingin menggunakan algoritma BFS atau DFS. Pengguna bisa menggunakan fitur bonus yaitu TSP dan pengguna juga bisa mengatur durasi tick program. Pengguna bisa menekan tombol *search* untuk memulai pencarian. Berikut merupakan tampilan dari program yang telah kami buat



Gambar 4.3.1 Tampilan awal program



Gambar 4.3.2 Tampilan akhir program

4.4 Hasil Pengujian

Pengujian Test Case 1

```

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

```

Gambar 4.4.1 Test Case 1

Treasure Hunt Solver

×

Input

Filename

sampel-1.txt

Browse

Algoritma

☒ BFS
☐ DFS
☐ TSP

Durasi Tick

50ms

Output

	Treasure		
			Treasure
Start			Treasure

Search

Route: R-U-U-D-R-R-D
Steps: 8
Nodes: 13
Time: 0 ms

Gambar 4.4.2 Pengujian Test Case 1 dengan BFS

Treasure Hunt Solver

×

Input

Filename

sampel-1.txt

Browse

Algoritma

☐ BFS
☒ DFS
☐ TSP

Durasi Tick

50ms

Output

	Treasure		
			Treasure
Start			Treasure

Search

Route: R-U-R-R-L-L-U-D-R-R-D
Steps: 12
Nodes: 14
Time: 1 ms

Gambar 4.4.3 Pengujian Test Case 1 dengan DFS

Pengujian Test Case 2

```
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
```

Gambar 4.4.4 Test Case 2

Treasure Hunt Solver

×

Input

Filename
sampel-2.txt Browse

Algoritma
☒ BFS
☐ DFS
☐ TSP

Durasi Tick
 50ms

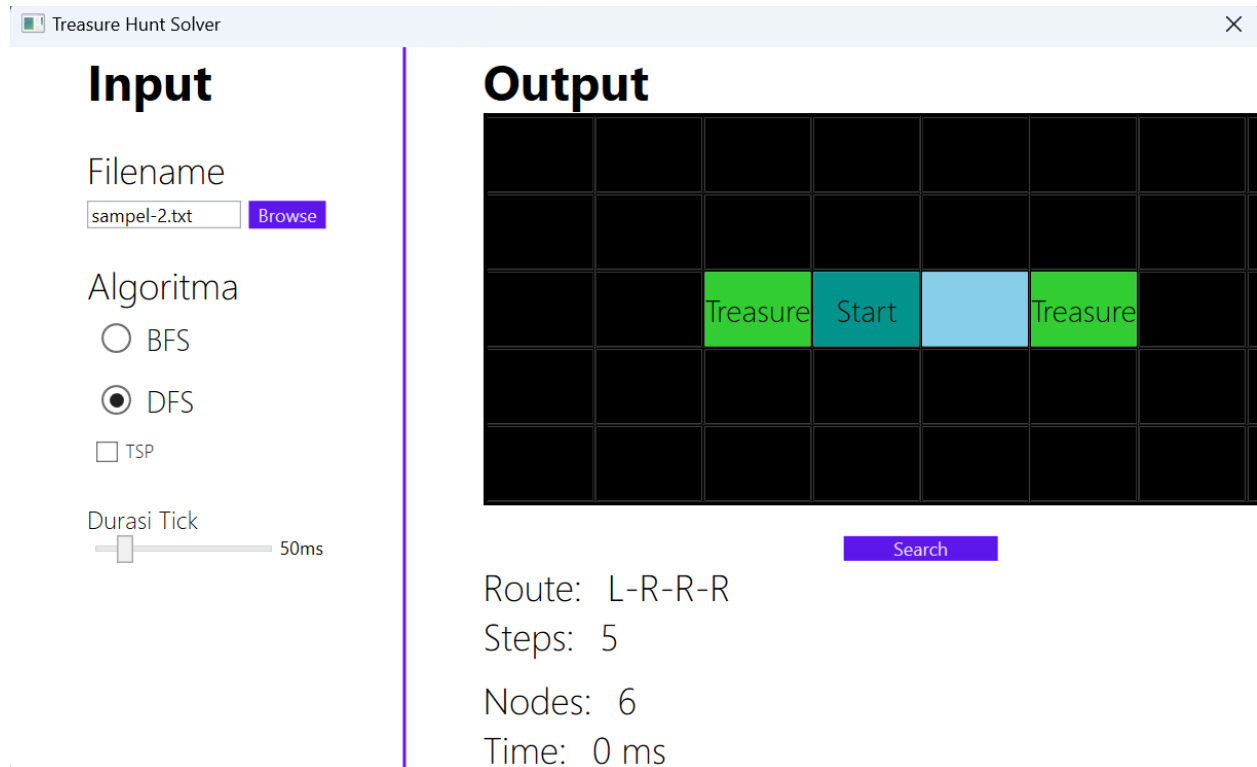
Output

		Treasure	Start		Treasure	

Search

Route: L-R-R-R
Steps: 5
Nodes: 5
Time: 0 ms

Gambar 4.4.5 Pengujian Test Case 2 dengan BFS



Gambar 4.4.6 Pengujian Test Case 2 dengan DFS

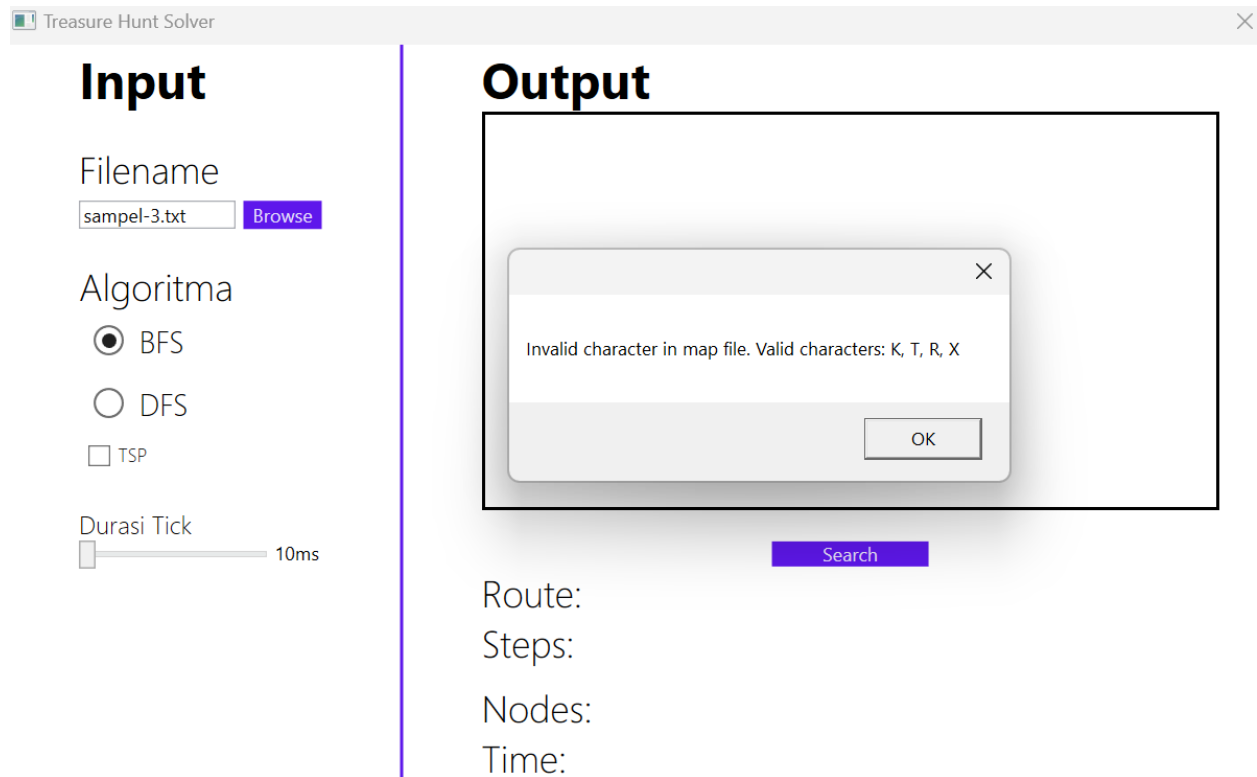
Pengujian Test Case 3

```

J A N G A N
L U P A C E
K Y A N G B
E G I N I Y

```

Gambar 4.4.7 Test Case 3

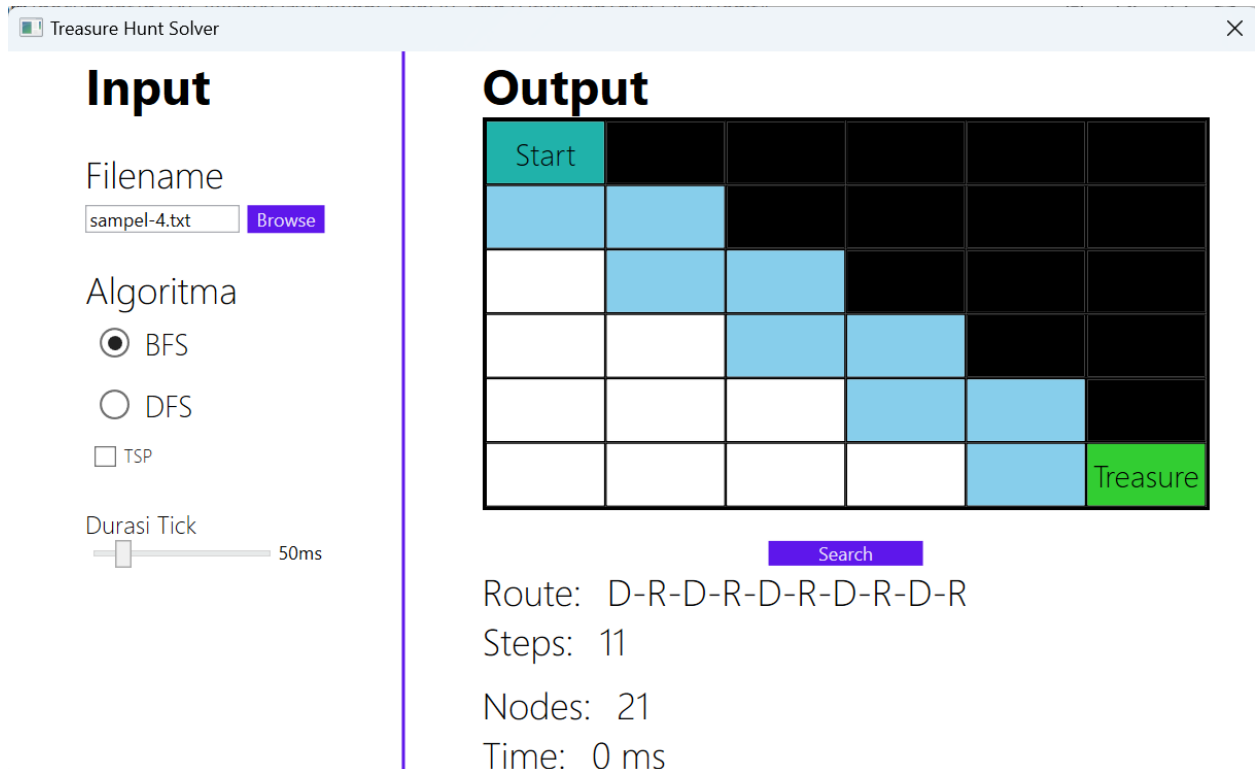


Gambar 4.4.8 Pengujian Test Case 3

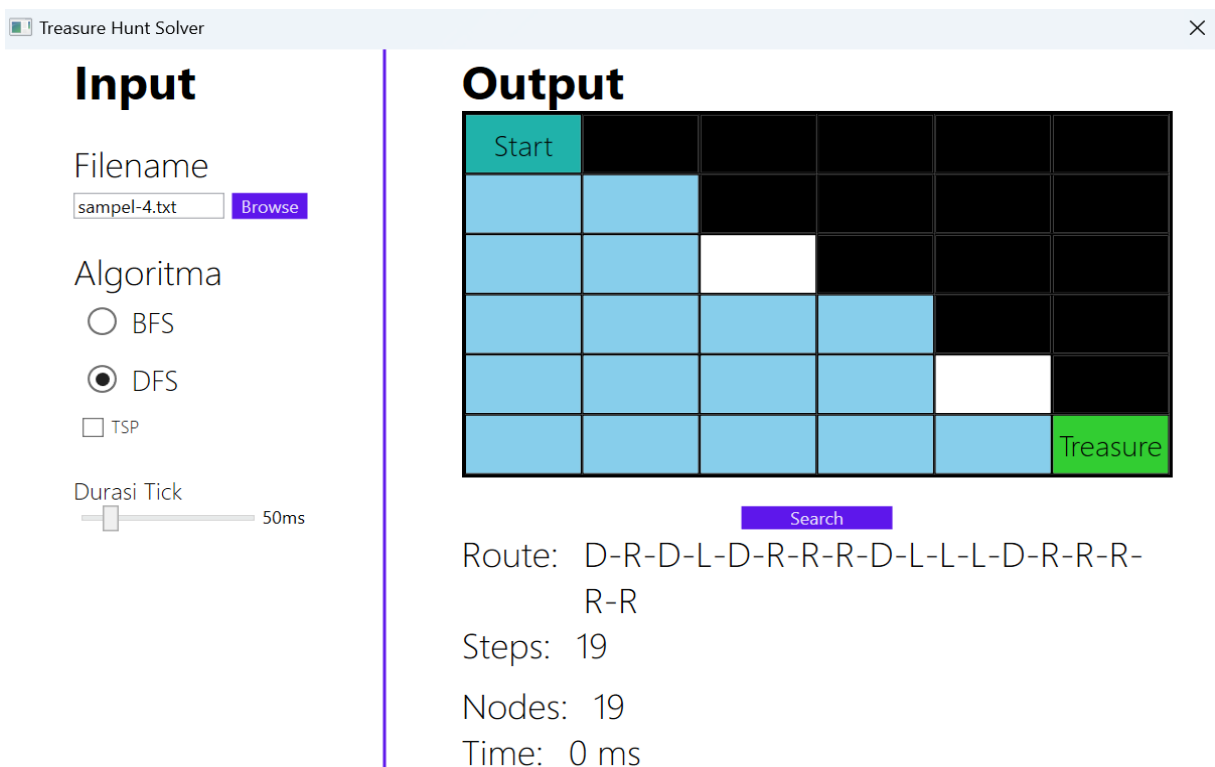
Pengujian Test Case 4

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

Gambar 4.4.9 Test Case 4



Gambar 4.4.10 Pengujian Test Case 4 dengan BFS



Gambar 4.4.11 Pengujian Test Case 4 dengan DFS

Pengujian Test Case 5

```
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
```

Gambar 4.4.12 Test Case 5

Treasure Hunt Solver

×

Input

Filename
sampel-5.txt Browse

Algoritma
☒ BFS
☐ DFS
☐ TSP

Durasi Tick
 50ms

Output

Start	Treasure		
	Treasure		
	Treasure		
	Treasure		
	Treasure		
	Treasure		
	Treasure		
	Treasure		
	Treasure		
	Treasure		

Search

Route: R-D-D-D-D-D-D-D-D-D-D-D

Steps: 12

Nodes: 32

Time: 0 ms

Gambar 4.4.13 Pengujian Test Case 5 dengan BFS

Bab 5

Kesimpulan dan Saran

Berdasarkan tugas ini kita dapat menyimpulkan bahwa algoritma *breath-first search* dan *depth-first search* dapat digunakan untuk menyelesaikan permasalahan *Maze Treasure Hunt*. Algoritma tersebut terbukti cukup efektif untuk menyelesaikan permasalahan ini. Hal tersebut dapat dilihat pada hasil pengujian, kedua algoritma tersebut sukses untuk menemukan *Treasure* pada *Maze*.

Saran-saran yang dapat kami berikan adalah:

- a. Spesifikasi tugas besar dapat dibuat lebih jelas
- b. Penulisan *pseudocode* terlihat kurang perlu karena program yang cukup panjang dan akan lebih mudah membaca program daripada *pseudocode*

Daftar Pustaka

Link Repository : https://github.com/ammарasyad/Tubes2_apaAjaDah

Link Youtube : <https://youtu.be/s098mZ4EuW4>

<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>

<http://csharp.net-informations.com/datagridview/csharp-datagridview-tutorial.htm>