

# Laporan Tugas Besar 2 IF2211 Strategi Algoritma Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt



Dibuat oleh:

Matthew Mahendra - 13521007

Muhamad Salman Hakim Alfarisi - 13521010

Hidayatullah Wildan Ghaly B. - 13521015

Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

# Daftar Isi

<b>1</b>	<b>Deskripsi Tugas</b>	<b>3</b>
1.1	Latar Belakang . . . . .	3
1.2	Deskripsi Tugas . . . . .	3
<b>2</b>	<b>Landasan Teori</b>	<b>5</b>
2.1	Graph Traversal . . . . .	5
2.1.1	Breadth-First Search . . . . .	5
2.1.2	Depth-First Search . . . . .	5
2.2	C# Desktop Application Development . . . . .	6
<b>3</b>	<b>Aplikasi Algoritma BFS dan DFS</b>	<b>7</b>
3.1	Langkah-Langkah Pemecahan Masalah . . . . .	7
3.1.1	Algoritma BFS . . . . .	7
3.1.2	Algoritma DFS . . . . .	7
3.2	Pemetaan Persoalan menjadi Elemen Algoritma BFS dan DFS . . . . .	7
3.3	Contoh Ilustrasi Pemecahan Masalah . . . . .	8
<b>4</b>	<b>Analisis Pemecahan Masalah</b>	<b>9</b>
4.1	Implementasi Program . . . . .	9
4.1.1	Implementasi Algoritma BFS . . . . .	9
4.1.2	Implementasi Algoritma DFS . . . . .	11
4.2	Struktur Data Program . . . . .	13
4.2.1	Vertex . . . . .	13
4.2.2	Map . . . . .	14
4.3	Tata Cara Penggunaan Program . . . . .	19
4.4	Hasil Pengujian . . . . .	20
4.4.1	Sampel 1 . . . . .	20
4.4.2	Sampel 2 . . . . .	21
4.4.3	Sampel 3 . . . . .	23
4.4.4	Sampel 4 . . . . .	23
4.4.5	Sampel 5 . . . . .	24
4.5	Analisis Desain Solusi . . . . .	26
<b>5</b>	<b>Kesimpulan dan Saran</b>	<b>27</b>
5.1	Simpulan . . . . .	27
5.2	Saran . . . . .	27
5.3	Refleksi . . . . .	27
5.4	Tanggapan Anggota . . . . .	28

<b>A Pranala Github</b>	<b>30</b>
<b>B Bonus Video Penjelasan</b>	<b>31</b>

# BAB 1

## Deskripsi Tugas

### 1.1 Latar Belakang

Tuan Krabs menemukan sebuah labirin distorsi terletak tepat di bawah Krusty Krab bernama El Doremi yang Ia yakini mempunyai sejumlah harta karun di dalamnya dan tentu saja Ia ingin mengambil harta karunnya. Dikarenakan labirinnya dapat mengalami distorsi, Tuan Krabs harus terus mengukur ukuran dari labirin tersebut. Oleh karena itu, Tuan Krabs banyak menghabiskan tenaga untuk melakukan hal tersebut sehingga Ia perlu memikirkan bagaimana caranya agar Ia dapat menelusuri labirin ini lalu memperoleh seluruh harta karun dengan mudah.

Setelah berpikir cukup lama, Tuan Krabs tiba-tiba mengingat bahwa ketika Ia berada pada kelas Strategi Algoritma-nya dulu, Ia ingat bahwa Ia dulu mempelajari algoritma BFS dan DFS sehingga Tuan Krabs menjadi yakin bahwa persoalan ini dapat diselesaikan menggunakan kedua algoritma tersebut. Akan tetapi, dikarenakan sudah lama tidak menyentuh algoritma, Tuan Krabs telah lupa bagaimana cara untuk menyelesaikan persoalan ini dan Tuan Krabs pun kebingungan. Tidak butuh waktu lama, Ia terpikirkan sebuah solusi yang brilian. Solusi tersebut adalah meminta mahasiswa yang saat ini sedang berada pada kelas Strategi Algoritma untuk menyelesaikan permasalahan ini.

### 1.2 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
- Grid halangan yang tidak dapat diakses

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.

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 menghandle kasus apabila tidak ditemukan dengan nama file tersebut.

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.

# BAB 2

## Landasan Teori

### 2.1 Graph Traversal

Graph traversal merupakan metode penelusuran simpul dalam sebuah graf. Proses yang dilakukan adalah mengunjungi simpul-simpul dengan urutan yang disesuaikan dengan urutan pengunjungan. Urutan pengunjungan yang dapat dilakukan adalah breadth-first search dan depth-first search.

#### 2.1.1 Breadth-First Search

Breadth-First Search atau BFS merupakan penelusuran graf secara melebar. Secara umum, algoritmanya adalah sebagai berikut,

1. Kunjungi simpul  $v$
2. Kunjungi simpul yang bertetangga dengan simpul  $v$
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang sudah dikunjungi hingga semua simpul sudah dikunjungi

Penelurusan menggunakan BFS umumnya menggunakan struktur data queue, dengan mencatat simpul-simpul yang bertetangga untuk diproses dengan queue.

#### 2.1.2 Depth-First Search

Depth-First Search atau DFS merupakan penelusuran graf secara mendalam. Secara umum, algoritmanya adalah sebagai berikut,

1. Kunjungi simpul  $v$
2. Kunjungi simpul  $w$  yang bertetangga dengan simpul  $v$
3. Ulangi DFS mulai dari simpul  $w$
4. Pada kondisi semua simpul dari simpul  $w$  sudah dikunjungi, maka lakukan pencarian runut balik untuk mencari cabang simpul yang dapat ditelusuri lagi
5. Proses dihentikan ketika tidak ada lagi simpul yang belum dikunjungi

Penelurusan menggunakan DFS umumnya menggunakan struktur data stack, dengan mencatat simpul-simpul yang bertetangga untuk diproses dengan stack.

## 2.2 C# Desktop Application Development

Bahasa pemrograman C# dapat dirancang secara manual maupun dengan bantuan framework. C# desktop application development menggunakan Visual Studio untuk merancang sebuah framework untuk mempermudah pembuatan aplikasi desktop.

Untuk pembuatan aplikasi dengan graphic user interface (GUI), digunakan framework Windows Forms.

# BAB 3

## Aplikasi Algoritma BFS dan DFS

### 3.1 Langkah-Langkah Pemecahan Masalah

#### 3.1.1 Algoritma BFS

Secara umum, implementasi BFS masih menggunakan queue dalam proses pencarinya. Dari titik awal peta, akan dilakukan pencarian dengan BFS. Setiap titik tetangga yang dapat dikunjungi dicatat arah pergerakannya menuju titik tersebut dan kemudian ditelusuri dari titik awal tersebut apakah sudah memenuhi hasil persoalan (menemukan semua harta karun).

Akan tetapi, untuk setiap titik yang dikunjungi, jika titik tersebut merupakan harta karun, maka daftar titik yang sudah dikunjungi akan dihapus dan diisi dengan titik tersebut. Dengan demikian, dapat dilakukan penelusuran dengan menginjak titik yang sudah dikunjungi sehingga dapat ditemukan satu jalur yang menyambung.

Prioritas gerakan pada program ini adalah, kanan, kiri, atas, dan bawah.

#### 3.1.2 Algoritma DFS

Algoritma DFS pada program ini adalah melakukan uji coba sesuai prioritas kanan, kiri, atas, dan bawah. Proses pencarian menggunakan stack of vertex dengan mencatat titik yang bertetangga dan mencatat titik yang sudah dikunjungi. Proses dilakukan hingga dicapai kondisi tidak dapat dilakukan pergerakan lagi karena semua titik sudah dikunjungi atau tidak dapat dikunjungi. Pada kondisi tidak dapat dilakukan pergerakan lagi, maka dilakukan runut balik hingga ada titik yang dapat dikunjungi lagi. Proses dilakukan hingga semua hadiah dapat ditemukan.

Prioritas gerakan pada program ini adalah, kanan, kiri, atas, dan bawah.

### 3.2 Pemetaan Persoalan menjadi Elemen Algoritma BFS dan DFS

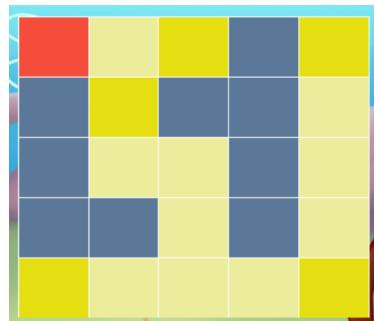
Persoalan Maze Treasure Hunt dapat dipetakan menjadi elemen algoritma BFS dan DFS sebagai berikut,

Nama Elemen	Deskripsi
Vertex	Vertex pada graf yang akan diselesaikan dengan bfs atau dfs berisi kumpulan titik-titik pada maze yang dapat dikunjungi atau merupakan sebuah titik yang berisi treasure
Edge	Edge pada graf yang akan diselesaikan dengan bfs atau dfs merupakan kumpulan arah yang dapat dikunjungi dari suatu titik tertentu. Edge ini tidak berarah sehingga memungkinkan untuk melakukan proses backtracking

Tabel 3.1: Tabel Pemetaan Persoalan menjadi Elemen Algoritma

### 3.3 Contoh Ilustrasi Pemecahan Masalah

Diberikan peta seperti gambar di bawah ini, dengan spesifikasi, warna kuning gelap adalah



Gambar 3.1: Contoh Peta

treasure/hadiah, warna biru tua tembok yang tidak dapat dilewati, warna merah titik awal, dan warna kuning muda tempat yang bisa dilewati.

Pemecahan dengan DFS, sesuai prioritas arah, akan menghasilkan gerakan ke kanan hingga hadiah pertama dicapai. Karena tidak dapat bergerak lagi, maka dilakukan proses runut balik hingga dapat ditemukan arah yang dapat dilewati lagi. Karena pencarian dilakukan terus menerus hingga tidak ada gerakan yang memungkinkan, maka pada baris paling bawah, arah pencarian akan mencari hingga ke ujung kanan peta, setelah itu baru kembali untuk mengambil hadiah di ujung kiri peta.

Pemecahan dengan BFS, sesuai prioritas arah, akan menghasilkan gerakan dengan mencari hadiah yang terdekat terlebih dahulu. Untuk baris paling bawah, karena hadiah lebih optimal didapatkan dengan bergerak ke kiri terlebih dahulu, maka akan bergerak ke kiri terlebih dahulu baru ke kanan. Secara tidak langsung hasil dari BFS adalah shortest path atau most minimum move.

# BAB 4

## Analisis Pemecahan Masalah

### 4.1 Implementasi Program

#### 4.1.1 Implementasi Algoritma BFS

Pseudocode untuk algoritma BFS yang digunakan adalah sebagai berikut,

```
1 procedure BFS(input: none, output: string of movements)
2 I.S. map terdefinisi, queue telah dibentuk
3 F.S. dihasilkan alur gerakan dengan pencarian BFS dari titik awal
      hingga akhir
4
5 Kamus Lokal
6 queue: Queue of Vertex
7 treasure, AlreadyVisited : Stack of Vertex
8 paths: string
9 nodesChecked: integer
10 move[]: char of movements (R, L, U, D)
11 start: Vertex
12 notValid: boolean
13
14 Algoritma
15 paths <- ""
16 queue.Enqueue(paths)
17 nodesChecked <- 0
18
19 while(alur gerakan belum menemukan solusi)
20     start <- ambil titik awal pada peta
21     paths <- queue.Dequeue()
22     i iterate 0..4
23         notValid <- false
24
25         foreach char move in paths + move[i]
26             depend on move:
27                 move = 'R'
28                     if (dari start tidak dapat bergerak ke kanan)
29                         then
30                             notValid <- true
31                             break
```

```

31         else
32             start.x <- start.x + 1
33             if (posisi start sekarang adalah treasure
34                 and treasure belum ada di stack Treasure)
35                 then
36                     AlreadyVisited.Clear()
37                     Treasure.Push(start)
38                     AlreadyVisited.Push(start)
39             move = 'L'
40             if (dari start tidak dapat bergerak ke kiri)
41                 then
42                     notValid <- true
43                     break
44             else
45                 start.x <- start.x - 1
46                 if (posisi start sekarang adalah treasure
47                     and treasure belum ada di stack Treasure)
48                     then
49                         AlreadyVisited.Clear()
50                         Treasure.Push(start)
51                         AlreadyVisited.Push(start)
52             move = 'U'
53             if (dari start tidak dapat bergerak ke atas)
54                 then
55                     notValid <- true
56                     break
57             else
58                 start.y <- start.y - 1
59                 if (posisi start sekarang adalah treasure
60                     and treasure belum ada di stack Treasure)
61                     then
62                         AlreadyVisited.Clear()
63                         Treasure.Push(start)
64                         AlreadyVisited.Push(start)
65             move = 'D'
66             if (dari start tidak dapat bergerak ke bawah)
67                 then
68                     notValid <- true
69                     break
70             else
71                 start.y <- start.y + 1
72                 if (posisi start sekarang adalah treasure
73                     and treasure belum ada di stack Treasure)
74                     then
75                         AlreadyVisited.Clear()
76                         Treasure.Push(start)
77                         AlreadyVisited.Push(start)

78             if (!notValid) then
79                 queue.Enqueue(paths + move[i])
80

```

```

71     nodesChecked <- nodesChecked + 1
72
73 nodesChecked <- nodesChecked - 1
74 -> paths

```

#### 4.1.2 Implementasi Algoritma DFS

Pseudocode untuk algoritma DFS yang digunakan adalah sebagai berikut,

```

1 procedure DFS(input: none, output: string of movements)
2 I.S. map terdefinisi, stack telah dibentuk
3 F.S. dihasilkan alur gerakan dengan pencarian DFS dari titik awal
      hingga akhir
4
5 Kamus Lokal
6 stack, visited, backtrack: stack of Vertex
7 path: stack of char
8 temp, startVertex, current: Vertex
9 nodesChecked, treasure, treasureFound: integer
10 signal, isSUS: boolean
11 paths: string
12
13 treasure <- map.getNumOfTreasure()
14 treasureFound <- 0
15 startVertex <- map.getStartingPoint()
16 temp <- startVertex
17
18 stack.Push(startVertex)
19 visited.Push(startVertex)
20 backtrack.Push(startVertex)
21
22 while(stack is not empty and treasureFound != treasure)
23     current <- stack.Pop()
24     backtrack.Push(current)
25     visited.Push(current)
26
27     { Prioritas pengisian dilakukan terbalik sesuai dengan sifat
      tipe data stack }
28     if(current adalah treasure) then
29         treasureFound <- treasureFound + 1
30
31     if(gerakan ke bawah dari current valid) then
32         stack.Push(titik di bawah current);
33         nodesChecked++;
34
35     if(gerakan ke atas dari current valid) then
36         stack.Push(titik di atas current);
37         nodesChecked++;
38
39     if(gerakan ke kiri dari current valid) then
40         stack.Push(titik di kiri current);

```

```
41     nodesChecked++;
42
43     if(gerakan ke kanan dari current valid) then
44         stack.Push(titik di bawah current);
45         nodesChecked++;
46
47     signal <- false
48
49     if(tidak ada gerakan lain yang valid) then
50         signal <- true
51
52     isSUS <- false
53
54 { PROSES BACKTRACKING }
55
56 if(temp != current) then
57     depend on (current, temp)
58     temp berada di kanan current: path.Push('R')
59     temp berada di kiri current: path.Push('L')
60     temp berada di atas current: path.Push('U')
61     temp berada di bawah current: path.Push('D')
62     else: Error
63 else
64     { DO NOTHING }
65
66 while (tidak ada gerakan lain yang valid and treasureFound <
       treasure)
67     temp <- current
68     current <- backtrack.Pop()
69
70     if (current.GetStatusTreasure() and !isSUS){
71         isSUS <- true
72     } else if (!isSUS) {
73         if (this.m.isBackTrack(current, this.m.getMap(), visited
                           )) then
74             path.Pop()
75     } else if (isSUS) {
76         depend on (current, temp)
77         temp berada di kanan current: path.Push('R')
78         temp berada di kiri current: path.Push('L')
79         temp berada di atas current: path.Push('U')
80         temp berada di bawah current: path.Push('D')
81         else: Error
82     }
83
84     if (!signal)
85         temp <- current
86         backtrack.Push(current)
87
88 {Kembalikan path secara terbalik}
89
```

90 | -> paths

## 4.2 Struktur Data Program

### 4.2.1 Vertex

Vertex adalah kelas untuk mendefinisikan setiap titik pada peta. Kelas ini terdiri dari atribut integer dan x dan y sebagai penanda koordinat pada matriks peta dan boolean isTreasure dan isAvailable sebagai penanda apakah suatu titik dapat dilewati atau tidak serta apakah suatu titik adalah hadiah atau bukan.

Representasi dalam bahasa pemrograman C# untuk kelas ini adalah sebagai berikut,

```
1  using System;
2
3  public class Vertex
4  {
5      /* Attributes */
6
7      /* Koordinat pada Peta */
8      public int x;
9      public int y;
10
11     /* isTreasure menandakan treasure: default false;
12      * isAvailable menandakan apakah vertex dapat diinjak:
13      * default false
14      */
15     private bool IsTreasure;
16     private bool IsAvailable;
17
18     public Vertex() { this.x = 0; this.y = 0; IsTreasure = false;
19         ; IsAvailable = false; }
20
21     public Vertex(int x, int y, bool t, bool a) { this.x = x;
22         this.y = y; this.IsTreasure = t; this.IsAvailable = a; }
23
24     public bool GetStatusTreasure() { return IsTreasure; }
25
26     public bool GetStatusMove() { return IsAvailable; }
27
28     public int getRow() { return x; }
29
30     public int getCol() { return y; }
31
32     public char getValue() { return IsTreasure ? 'T' : ' '; }
33
34     public void disableTreasure() { IsTreasure = false; }
35
36     /* Edit Contains */
37     public override bool Equals(object obj)
38     {
39
40         if (obj == null) return false;
41
42         if (typeof(obj) != typeof(this)) return false;
43
44         Vertex vertex = (Vertex)obj;
45
46         if (vertex.x != this.x || vertex.y != this.y) return false;
47
48         if (vertex.IsTreasure != IsTreasure) return false;
49
50         return true;
51     }
52
53 }
```

```

36     if(obj == null || GetType() != obj.GetType())
37     {
38         return false;
39     }
40
41     Vertex v = (Vertex)obj;
42     return (x == v.x && y == v.y);
43 }
44
45 public override int GetHashCode()
46 {
47     return (x + y).GetHashCode();
48 }
49 }
```

#### 4.2.2 Map

Map adalah kelas untuk mendefinisikan peta dengan matriks. Kelas terdiri dari getter dan matriks of Vertex dan of char.

Representasi dalam bahasa pemrograman C# untuk kelas ini adalah sebagai berikut,

```

1 using System;
2
3 public class Map
4 {
5     /* Ukuran Peta: x dan y */
6     public static int MapX, MapY;
7     public int treasureCount = 0;
8
9     /* Gerakan yang mungkin
10      * Up, Right, Down, Left */
11     int[] MoveX = { 0, 1, 0, -1 };
12     int[] MoveY = { -1, 0, 1, 0 };
13
14     /* Ini buat DFS BFS */
15     Vertex[,] Buffer;
16     char[,] map;
17
18     /* MAP Matrix of what */
19     /* x dan y posisi, map peta, */
20
21     /* Validator */
22     public bool isUpValid(Vertex point, char[,] map, Stack<
23         Vertex> stack)
24     {
25         if (point.y + MoveY[0] < MapY && point.y + MoveY[0]
26             >= 0 && !stack.Contains(this.getUp(point)))
27         {
28             return (map[point.y+MoveY[0],point.x] != 'X'
29             );
30         } else {
```

```

28             return false;
29         }
30     }
31
32     public bool isDownValid(Vertex point, char[,] map, Stack<
33         Vertex> stack)
34     {
35         if (point.y + MoveY[2] < MapY && point.y + MoveY[2]
36             >= 0 && !stack.Contains(this.getDown(point)))
37         {
38             return (map[point.y + MoveY[2], point.x] != 'X');
39         }
40     }
41     public bool isRightValid(Vertex point, char[,] map, Stack<
42         Vertex> stack)
43     {
44         if (point.x + MoveX[1] < MapX && point.x + MoveX[1]
45             >= 0 && !stack.Contains(this.getRight(point)))
46         {
47             return (map[point.y, point.x + MoveX[1]] != 'X');
48         }
49     }
50     public bool isLeftValid(Vertex point, char[,] map, Stack<
51         Vertex> stack)
52     {
53         if (point.x + MoveX[3] < MapX && point.x + MoveX[3]
54             >= 0 && !stack.Contains(this.getLeft(point)))
55         {
56             return (map[point.y, point.x + MoveX[3]] != 'X');
57         }
58     }
59     public bool isBackTrack(Vertex point, char[,] map, Stack<
60         Vertex> stack)
61     {
62         if (!isValid(point, map, stack) && !isDownValid(
63             point, map, stack) && !isLeftValid(point, map,
64             stack) && !isRightValid(point, map, stack))
65         {
66             return true;
67         }
68         else
69         {
70             return false;
71         }
72     }

```

```
67    }
68
69    public bool isStartingPoint(Vertex point, char[,] map)
70    {
71        return (map[point.y, point.x] == 'K');
72    }
73
74    public Vertex getStartingPoint (char[,] map)
75    {
76        for (int i = 0; i < MapY; i++) {
77            for (int j = 0; j < MapX; j++) {
78                if (map[i, j] == 'K') {
79                    return new Vertex(j, i,
80                                     false, true);
81                }
82            }
83            return new Vertex(0, 0, false, false);
84        }
85
86        public Map()
87        {
88            MapX = 0;
89            MapY = 0;
90            map = new char[MapY, MapX];
91            Buffer = new Vertex[MapY, MapX];
92        }
93
94        public int getTreasureCount()
95        {
96            return treasureCount;
97        }
98        public Map(string file)
99        {
100            string[] lines = System.IO.File.ReadAllLines(file);
101            int countY = 0;
102            int countX = 1;
103            foreach(string line in lines){
104                countY++;
105
106                if(countY == 1) {
107                    foreach(char word in line){
108                        if(word == ' ') {
109                            countX++;
110                        }
111                    }
112                }
113            }
114            MapX = countX;
115            MapY = countY;
116        }
```

```

117
118     map = new char[countY, countX];
119     for (int y = 0; y < countY; y++)
120     {
121         string line = lines[y];
122         int j = 0;
123         for (int x = 0; x < line.Length; x+=2)
124         {
125             char c = line[x];
126             map[y, j] = c;
127             j++;
128         }
129     }
130
131     Buffer = new Vertex[countY, countX];
132     Console.Write(countX + " " + countY + "\n");
133     for (int i = 0; i < countY; i++)
134     {
135         for (int j = 0; j < countX; j++)
136         {
137             if (map[i, j] == 'X')
138                 Buffer[i, j] = new Vertex(j,
139                                         i, false, false);
140             else if (map[i, j] == 'T')
141                 Buffer[i, j] = new Vertex(j,
142                                         i, true, true);
143             treasureCount++;
144         }
145     }
146
147
148     // Print the matrix for testing purposes
149     for (int y = 0; y < countY; y++)
150     {
151         for (int x = 0; x < countX; x++)
152         {
153             Console.Write(map[y, x] + " ");
154         }
155         Console.WriteLine();
156     }
157     Console.Write("Treasure_Count:" + treasureCount + "
158 \n");
159
160     public char[,] getMap()
161     {
162         return map;
163     }

```

```
164
165     public Vertex getVertex(int x, int y)
166     {
167         return Buffer[y, x];
168     }
169
170     public Vertex getRight(Vertex point)
171     {
172         return Buffer[point.y, point.x + 1];
173     }
174
175     public Vertex getLeft(Vertex point)
176     {
177         return Buffer[point.y, point.x - 1];
178     }
179
180     public Vertex getUp(Vertex point)
181     {
182         return Buffer[point.y - 1, point.x];
183     }
184
185     public Vertex getDown(Vertex point)
186     {
187         return Buffer[point.y + 1, point.x];
188     }
189
190     public Vertex getVertex(Vertex point)
191     {
192         return Buffer[point.y, point.x];
193     }
194
195     public void setVertex(Vertex point)
196     {
197         Buffer[point.y, point.x] = point;
198     }
199
200     public bool isRight(Vertex p1, Vertex p2)
201     {
202         return (p1.y == p2.y && p1.x + 1 == p2.x);
203     }
204
205     public bool isLeft(Vertex p1, Vertex p2)
206     {
207         return (p1.y == p2.y && p1.x - 1 == p2.x);
208     }
209
210     public bool isUp(Vertex p1, Vertex p2)
211     {
212         return (p1.y - 1 == p2.y && p1.x == p2.x);
213     }
214
```

```
215     public bool isDown(Vertex p1, Vertex p2)
216     {
217         return (p1.y + 1 == p2.y && p1.x == p2.x);
218     }
219 }
```

## 4.3 Tata Cara Penggunaan Program

Untuk menjalankan program ini, unduh program dari repositori Github yang dilampirkan serta lakukan instalasi .NET minimal versi 6.0. Setelah itu, lakukan langkah-langkah di bawah ini,

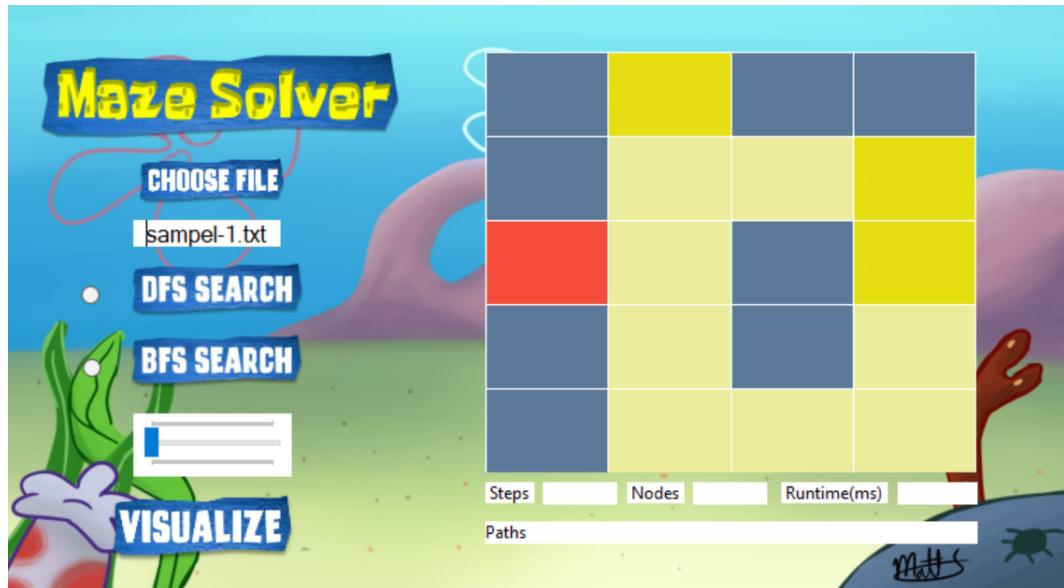
- Pada terminal, masukkan ‘dotnet run’ lalu program akan dijalankan
- Pilih file konfigurasi yang telah dimasukkan pada folder ‘test’. File akan otomatis melakukan visualisasi
- Pilih metode pencarian yang ingin digunakan (DFS atau BFS). Ada dua metode DFS, DFS dengan menampilkan proses runut balik dan DFS yang tidak menampilkan proses runut balik (langsung hasilnya). Prioritas arah adalah Kanan, Kiri, Atas, Bawah.
- Pilih kecepatan penelusuran menggunakan slider. Semakin ke kiri artinya lebih cepat, ke kanan lebih lambat

Program akan menampilkan alur gerakan, runtime, jumlah titik yang diperiksa, dan jumlah langkah yang diperlukan hingga menuju hasil.

## 4.4 Hasil Pengujian

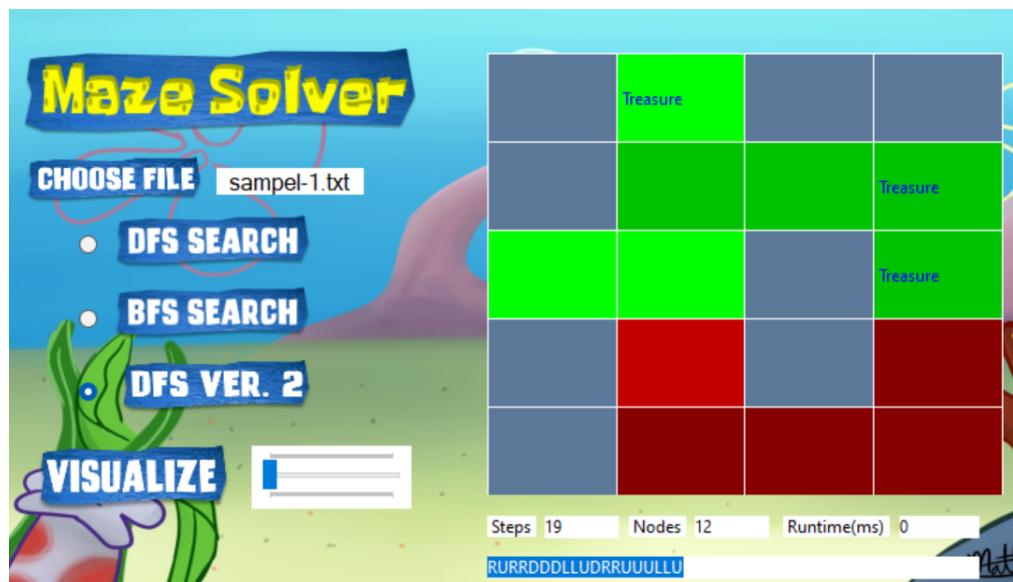
Pengujian menggunakan kasus tes yang diberikan oleh Asisten IF2211 Strategi Algoritma Tahun Akademik 2022-2023.

### 4.4.1 Sampel 1



Gambar 4.1: Peta untuk Sampel 1

DFS



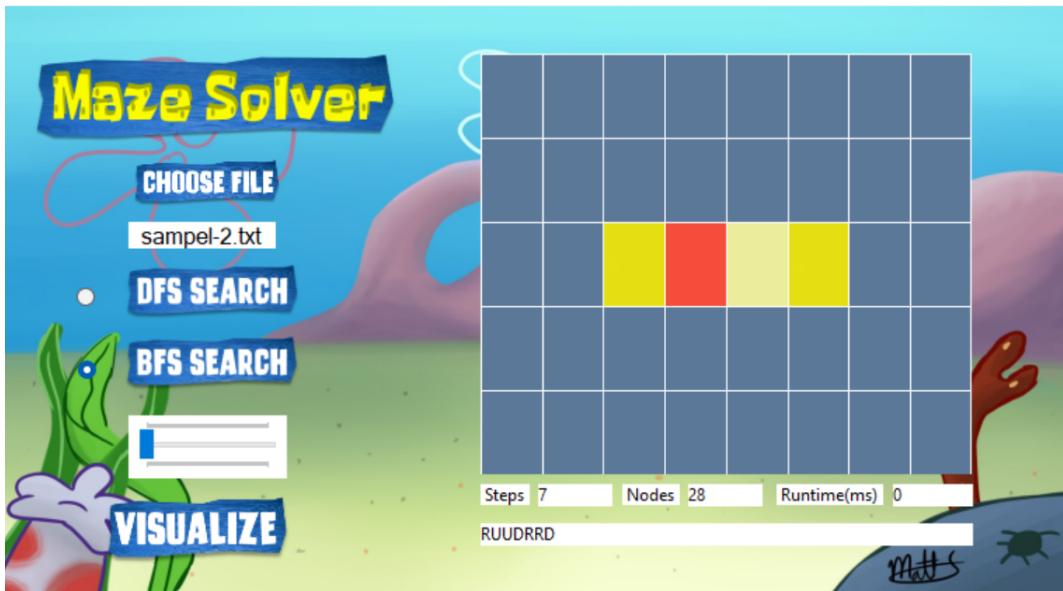
Gambar 4.2: Hasil DFS untuk Sampel 1

## BFS



Gambar 4.3: Hasil BFS untuk Sampel 1

### 4.4.2 Sampel 2



Gambar 4.4: Peta untuk Sampel 2

## DFS



Gambar 4.5: DFS untuk Sampel 2

## BFS



Gambar 4.6: BFS untuk Sampel 2

#### 4.4.3 Sampel 3

File .txt untuk sampel 3 sengaja dibuat salah dalam konfigurasinya sehingga tidak dapat ditampilkan.



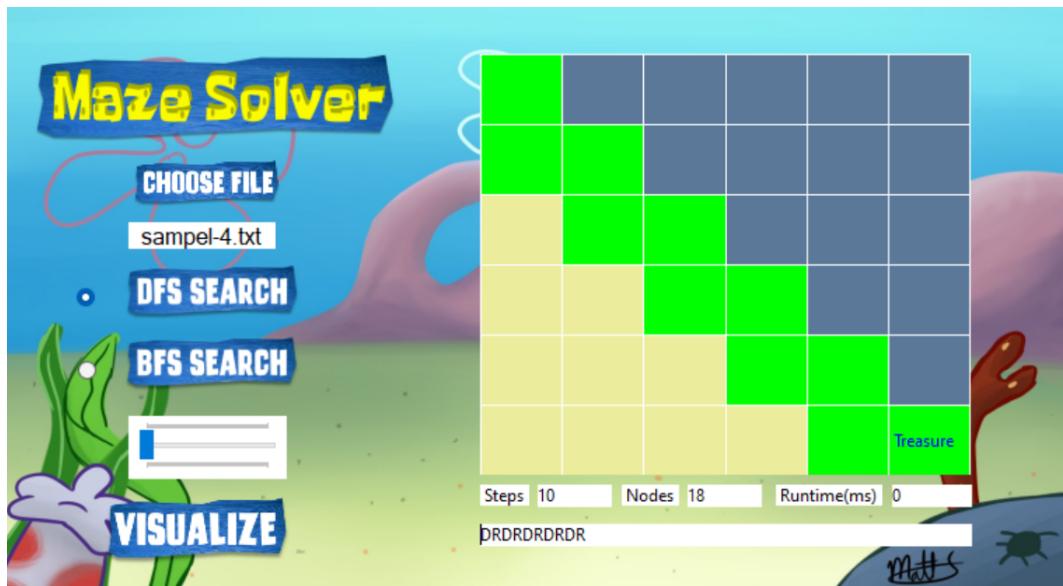
Gambar 4.7: Tampilan Error untuk Sampel 3

#### 4.4.4 Sampel 4



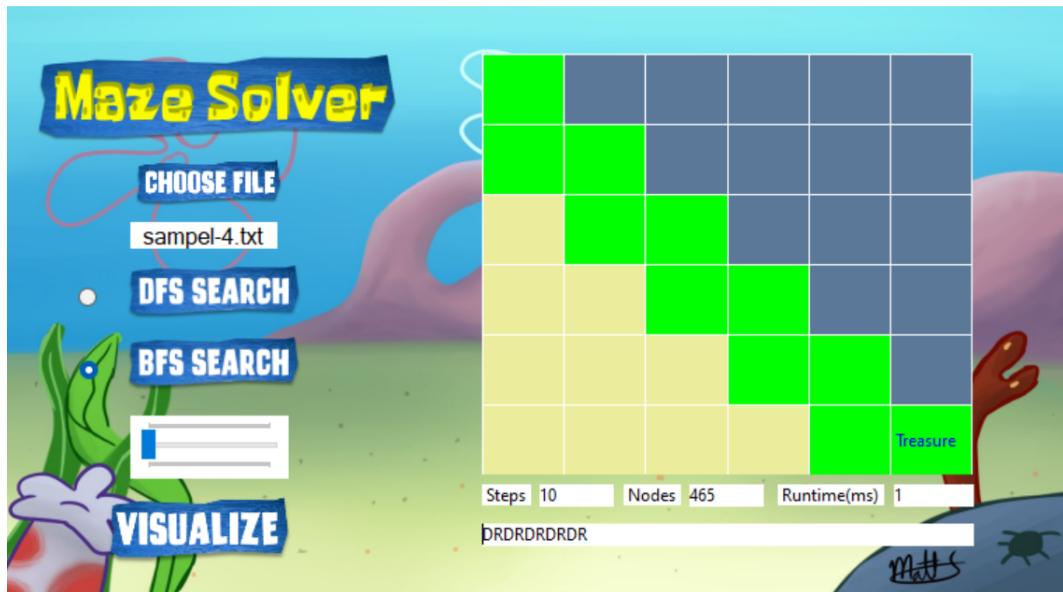
Gambar 4.8: Hasil DFS untuk Sampel 4

## DFS



Gambar 4.9: Hasil DFS untuk Sampel 4

## BFS



Gambar 4.10: Hasil BFS untuk Sampel 4

### 4.4.5 Sampel 5

## DFS



Gambar 4.11: Hasil DFS untuk Sampel 5



Gambar 4.12: Hasil DFS untuk Sampel 5

## BFS



Gambar 4.13: Hasil BFS untuk Sampel 5

## 4.5 Analisis Desain Solusi

Penyelesaian dengan DFS memeriksa lebih sedikit node. Akan tetapi, langkah yang dibutuhkan menjadi lebih banyak dan cenderung menjadi kurang efektif dalam langkah bergeraknya. Hal ini disebabkan oleh sifat DFS itu sendiri yang mencari sampai tidak ada langkah yang dapat dilakukan lagi, didukung dengan sifat first in last out dari stack. Menggunakan DFS pula, dapat dihasilkan visualisasi menggunakan backtracking (secara penelusuran) maupun hasil yang straight forward.

Penyelesaian dengan BFS menghasilkan langkah yang lebih efektif dan langkah yang membutuhkan jumlah gerakan yang lebih sedikit. Akan tetapi, kurang efektif karena memeriksa node yang lebih banyak. Hal ini disebabkan dari sifat queue yang bersifat first in first out ditambah dengan proses penghapusan queue visited ketika menemui sebuah treasure untuk menghasilkan langkah yang searah dari awal hingga akhir membuat jumlah titik yang perlu diperiksa menjadi lebih banyak.

# BAB 5

## Kesimpulan dan Saran

### 5.1 Simpulan

Dari tugas besar ini, disimpulkan hal-hal sebagai berikut,

1. DFS dan BFS dapat diterapkan pada masalah pencarian arah
2. DFS dan BFS dapat dimodifikasi untuk mendapatkan solusi sesuai yang dibutuhkan, tetapi tetap taat pada aturan yang berlaku dalam proses pencarian solusinya
3. Algoritma dengan BFS dapat menghasilkan least moves required dalam mencapai solusi
4. Algoritma dengan BFS menggunakan struktur data queue, sedangkan pada DFS menggunakan struktur data stack

### 5.2 Saran

Saran untuk perbaikan pada program ini adalah sebagai berikut,

1. BFS dapat dibuat lebih efektif dengan menggunakan priority queue untuk mempercepat pencarian
2. GUI dapat diperindah
3. Untuk BFS dan DFS, dapat menggunakan fungsi untuk mengecek apakah gerakan sudah valid atau belum agar lebih efektif

### 5.3 Refleksi

Melalui tugas besar ini, kelompok belajar untuk berpikir kreatif dan bekerja sama dalam kelompok. Disamping itu, juga belajar untuk dapat beradaptasi dengan lingkungan yang baru, khususnya dalam memahami bahasa pemrograman C# yang digunakan dalam pembuatan tugas besar ini.

## 5.4 Tanggapan Anggota

- Matthew Mahendra: Lumayan bisa belajar bahasa pemrograman C# sekaligus juga explore GUI. Menantang juga nyari solusinya supaya bisa tampilin jalurnya tapi juga tetep DFS dan BFS
- Muhammad Salman Hakim Alfarisi: Tubesnya seru, nyobain bahasa baru, bisa bikin gui, lebih paham dfs bfs
- Hidayatullah Wildan Ghaly B.: Tubesnya lumayan menantang, seru bisa eksplor bahasa baru dengan temen sekelompok :D

# Daftar Pustaka

- [1] R. Munir and N. U. Maulidevi, “Breadth/depth first search,” 2021.
- [2] A. Blades, “Solving mazes with depth-first search,” <https://medium.com/swlh/solving-mazes-with-depth-first-search-e315771317ae>, 2020.
- [3] Tim, “Python path finding tutorial - breadth first search algorithm,” <https://www.youtube.com/watch?v=hettiSrJjM4>, 2019.

# **Lampiran A**

## **Pranala Github**

Program dapat diakses pada repositori Github di tautan berikut: [https://github.com/MHEN2606/Tubes2\\_theMazeCoder](https://github.com/MHEN2606/Tubes2_theMazeCoder)

## **Lampiran B**

### **Bonus Video Penjelasan**

Penjelasan program dengan video dapat diakses pada tautan berikut: [https://youtu.be/\\_LGXkdTv4k](https://youtu.be/_LGXkdTv4k)