

Laporan Tugas Besar 2 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan
Maze Treasure Hunt



Disusun oleh:

Kelompok CrabRave

Muhammad Bangkit Dwi Cahyono 13521055

Louis Caesa Kesuma 13521069

Addin Munawwar Yusuf 13521085

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

Daftar Isi

Daftar Isi	2
Bab 1	
Deskripsi Tugas	4
1.1. Deskripsi Tugas	4
1.2. Spesifikasi GUI	6
1.3. Spesifikasi Wajib	8
Bab 2	
Landasan Teori	9
2.1. Dasar Teori	9
2.1.1. Graph Traversal	9
2.1.2. Breadth First Search (BFS)	9
2.1.3. Depth First Search (DFS)	10
2.2. Pengembangan Aplikasi Desktop	11
2.2.1. Bahasa Pemrograman C#	11
2.2.2. .NET Framework dengan Windows Presentation Foundation (WPF)	11
Bab 3	
Analisis Pemecahan Masalah	13
3.1. Langkah-Langkah Pemecahan Masalah	13
3.2. Mapping Persoalan ke Elemen Algoritma BFS dan DFS	13
3.3. Contoh Ilustrasi Kasus Lain	14
Bab 4	
Implementasi dan Pengujian	15
4.1. Implementasi Program (Pseudocode Program Utama)	15
4.2.1. Implementasi BFS	15
4.2.2. Implementasi DFS	21
4.2. Penjelasan Struktur Data	28
4.2.1. Node	28
4.2.2. Map	29
4.2.3. List	29
4.2.4. Queue	29
4.2.5. Stack	30
4.3. Penjelasan Tata Cara Penggunaan Program	30
4.4. Hasil Pengujian	32
4.4.1. Test Case 1	32
4.4.2. Test Case 2	33
4.4.3. Test Case 3	35
4.4.4. Test Case 4	37

4.4.5. Test Case 5	39
4.5. Analisis Desain Solusi Algoritma BFS dan DFS	41
Bab 5	
Kesimpulan dan Saran	43
5.1. Kesimpulan	43
5.2. Saran	43
5.3. Refleksi	44
5.4. Tanggapan	44
Link Penting	45
Daftar Pustaka	46

Bab 1

Deskripsi Tugas

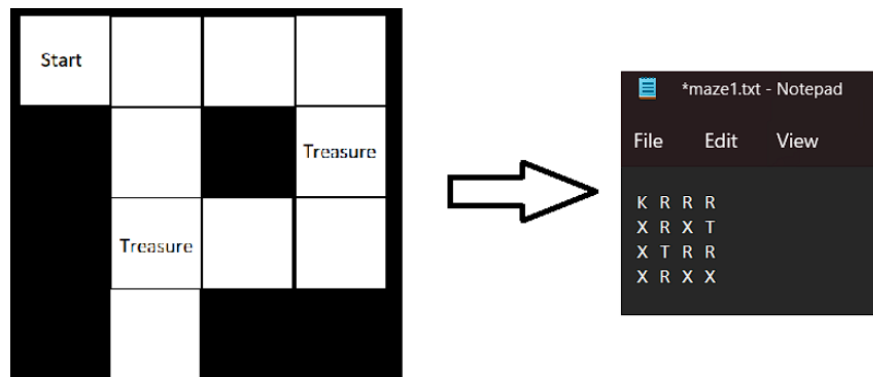
1.1. Deskripsi Tugas

Dalam tugas besar ini, kami diminta untuk membangun sebuah aplikasi dengan GUI sederhana yang dapat mengimplementasikan BFS dan DFS untuk mendapatkan rute untuk 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 untuk 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

Berikut adalah contoh file input dari program.

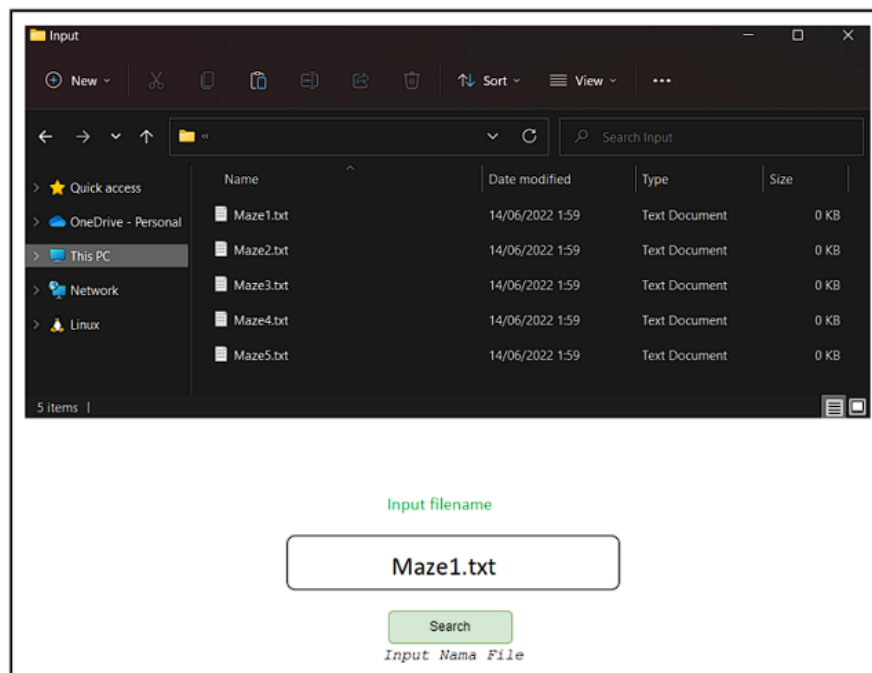


Gambar 1.1.1 Ilustrasi input file maze (*.txt)

Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), kami 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. Kami 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.

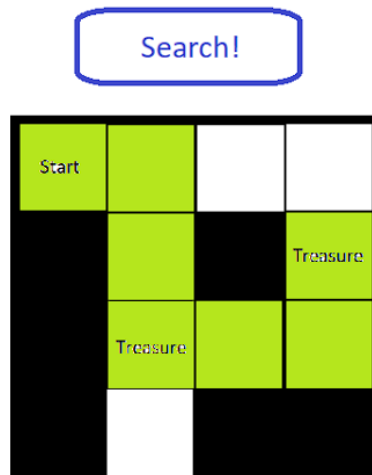
Berikut adalah contoh input aplikasi.



Gambar 1.1.2 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.

Berikut adalah contoh output aplikasi.

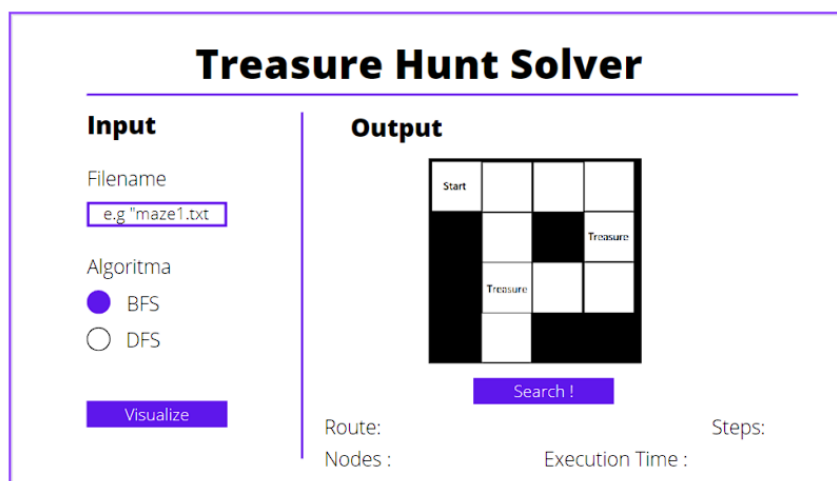


Gambar 3 Contoh output program untuk Gambar 1

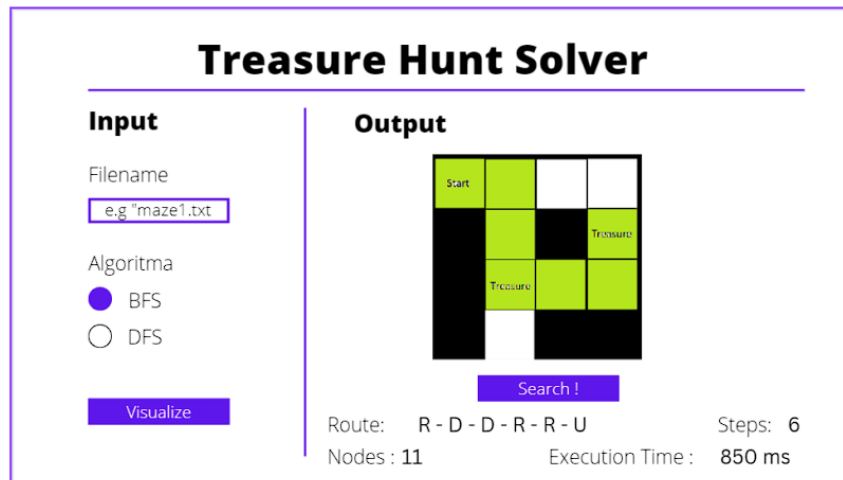
Setelah program melakukan pembacaan input, program akan memvisualisasikan gridnya terlebih dahulu tanpa pemberian rute solusi. Hal tersebut dilakukan agar pengguna dapat mengerjakan terlebih dahulu treasure hunt secara manual jika diinginkan. Kemudian, program menyediakan tombol solve untuk mengeksekusi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

1.2. Spesifikasi GUI

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



Gambar 1.2.1 Contoh output program untuk Gambar 1



Gambar 1.2.2 Contoh output program untuk Gambar 1

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.

Berikut spesifikasi GUI secara lengkap.

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 kreatif mungkin asalkan memuat 5 (7 jika mengerjakan bonus) spesifikasi di atas.

1.3. Spesifikasi Wajib

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.

Bab 2

Landasan Teori

2.1. Dasar Teori

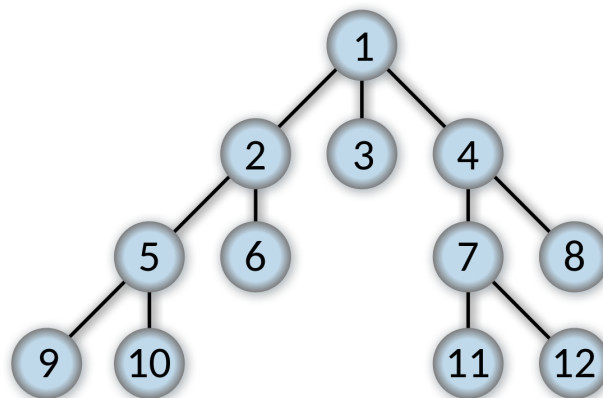
2.1.1. Graph Traversal

Graph traversal adalah algoritma penelusuran graf yang mengunjungi setiap simpul (*node*) secara sistematis. Algoritma yang digunakan pada *graph traversal* sangatlah beragam, mulai dari BFS, DFS, dan lain-lain.

Untuk menghindari pencarian simpul secara terus-terusan atau *infinite loop*, maka setiap simpul harus dicatat, apakah belum dikunjungi, sedang dikunjungi, atau sudah dikunjungi.

2.1.2. Breadth First Search (BFS)

BFS adalah salah satu dari banyak algoritma pencarian rute graf yang populer. BFS adalah algoritma yang melakukan pencarian per-tingkat (*level*) pada sebuah graf. BFS melakukan penelusuran secara merata terhadap setiap *node* pada graf. Keuntungan dari BFS adalah solusi yang dihasilkan biasanya lebih optimal dibandingkan hasil pencarian algoritma DFS, dikarenakan proses pencarian solusi dari algoritma BFS yang melakukan penelusuran secara menyeluruh dan satu per satu pada graf. Kekurangan dari algoritma ini adalah lamanya proses pencarian solusi dikarenakan algoritmanya memprioritaskan pola pencaharian yang merata terhadap setiap *node*.

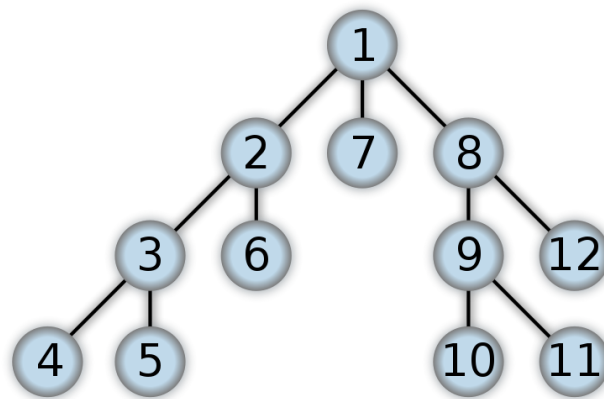


Gambar 2.1.2.1 Ilustrasi BFS

Dari *Gambar 2.1.2* dapat dilihat bahwa alur BFS menyebar (*breadth*) ditandai dengan nomor 1 hingga 12. Algoritmanya menjadi loncat-loncat simpul karena setelah memproses simpul anak, akan mengecek simpul saudara yang lainnya terlebih dahulu (mencari yang setara/ selevel).

2.1.3. Depth First Search (DFS)

DFS adalah salah satu algoritma pencarian rute graf yang populer. Keunikan DFS adalah algoritma pencariannya yang menjelajahi suatu jalan (*path*) tertentu sampai jalan tersebut sudah tidak bisa dijelajahi lagi. Jika sudah tidak bisa dijelajahi lagi, maka algoritmanya akan melakukan *backtracking* sambil mengecek apakah ada jalan yang belum ditelusuri oleh algoritma. Kekurangan dari algoritma DFS adalah prosesnya yang akan memakan waktu yang sangat lama walaupun solusi yang diinginkan tergolong dekat dengan posisi awal, namun dikarenakan adanya prioritas penelusuran jalan maka bisa jadi solusi yang terdekat tersebut akan dilewati terlebih dahulu.



Gambar 2.1.3.1 Ilustrasi DFS

Dari *Gambar 2.1.3* dapat dilihat bahwa alur DFS mendalam (*depth*) ditandai dengan nomor 1 hingga 12. Jika sudah mentok, maka ada proses *backtracking* ke parent sebelum-sebelumnya.

2.2. Pengembangan Aplikasi Desktop

2.2.1. Bahasa Pemrograman C#

C# atau dibaca C-Sharp adalah sebuah bahasa pemrograman yang dikembangkan oleh Microsoft dan berjalan di *framework* .NET. C# umumnya digunakan pada *web development*, aplikasi *desktop*, aplikasi *mobile*, *game*, dan sebagainya.

Berikut adalah beberapa kegunaan dari bahasa pemrograman C#.

1. Membuat Aplikasi Windows

C# dirancang untuk memungkinkan para pengembang membuat aplikasi desktop Windows yang efisien dan andal. Oleh karena itu, banyak perangkat lunak desktop Windows populer seperti Microsoft Office, Visual Studio, dan Adobe Photoshop ditulis dengan menggunakan C#.

2. Pengembangan Aplikasi Web

C# digunakan dalam pengembangan aplikasi web seperti ASP.NET, ASP.NET Core, dan Blazor. C# dapat digunakan untuk membuat berbagai jenis aplikasi web, mulai dari situs web sederhana hingga aplikasi web besar yang kompleks.

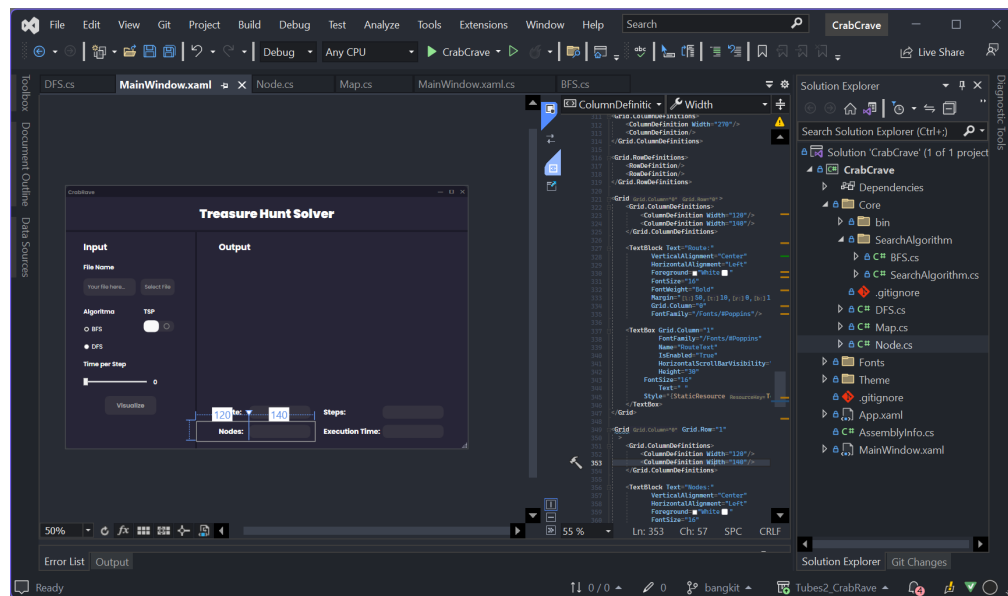
3. Membuat Aplikasi Mobile

C# dapat digunakan untuk membuat aplikasi mobile di berbagai platform seperti iOS dan Android. Untuk platform Android, C# digunakan dalam pengembangan aplikasi Android dengan menggunakan Xamarin, sedangkan untuk platform iOS, C# digunakan dalam pengembangan aplikasi iOS dengan menggunakan Xamarin.iOS.

2.2.2. .NET Framework dengan Windows Presentation Foundation (WPF)

.NET Framework atau dibaca dot net framework adalah perangkat lunak kerangka kerja yang dibentuk oleh Microsoft untuk memfasilitasi segala kebutuhan aplikasi yang sedang berjalan di Windows. Software ini menyediakan dua komponen utama: Common Language Runtime (CLR), untuk menangani aplikasi yang sedang berjalan, dan Framework Class Library (FCL), pustaka kode terkelola untuk melakukan pemrograman pada aplikasi.

Pada tugas kali ini, *framework* yang kami gunakan untuk membuat aplikasi desktop adalah WPF. *Windows Presentation Foundation* (WPF) adalah sebuah subsistem grafis yang dapat digunakan untuk membuat antarmuka pengguna pada aplikasi berbasis *Windows*. WPF juga menyediakan kemampuan untuk membuat aplikasi dengan tampilan 3D, animasi, dan efek visual yang kaya, serta integrasi dengan teknologi multimedia seperti video dan audio. WPF juga merupakan bagian dari *framework* .NET.



Gambar 2.2.2.1 Tampilan pengembangan menggunakan WPF

Bahasa pemrograman yang umumnya digunakan dalam pengembangan aplikasi WPF adalah C# dan VB.NET. Namun, untuk mendefinisikan antarmuka pengguna, WPF menggunakan XAML (Extensible Application Markup Language) yang memungkinkan pengembang untuk mendefinisikan antarmuka pengguna secara deklaratif, dengan menggunakan sintaks XML. XAML juga memungkinkan desainer antarmuka pengguna dan pengembang untuk bekerja secara terpisah, sehingga memungkinkan kolaborasi yang lebih baik dalam pengembangan aplikasi. Oleh karena itu, kombinasi antara C# dan XAML sering digunakan dalam pengembangan aplikasi WPF.

Bab 3

Analisis Pemecahan Masalah

3.1. Langkah-Langkah Pemecahan Masalah

Dalam menyelesaikan permasalahan ini, telah didekomposisi beberapa permasalahan sebagai langkah-langkah berikut ini.

1. Memahami dan mempelajari struktur data yang akan digunakan untuk implementasi penyelesaian persoalan.
2. Memahami konsep dari algoritma traversal BFS, DFS, dan juga TSP (berbasis BFS dan DFS).
3. Mempelajari *syntax* bahasa pemrograman C# dan .NET Framework (WPF, XAML) khususnya dalam mengakses file, membaca file, implementasi matrix yang berperilaku seperti graf dalam C#, asynchronous task, dan bagaimana caranya agar dapat tersambung dengan GUI.
4. Membuat Graphical User Interface (GUI) menggunakan .NET Framework pada Ms Visual Studio (WPF).
5. Memetakan setiap input menjadi elemen-elemen pada DFS dan BFS.
6. Mengimplementasikan algoritma BFS dan DFS pada pencarian rute pada peta persoalan.
7. Membuat visualisasi dan animasi pada GUI.
8. Menghubungkan seluruh program dengan GUI yang sudah dibuat.

3.2. Mapping Persoalan ke Elemen Algoritma BFS dan DFS

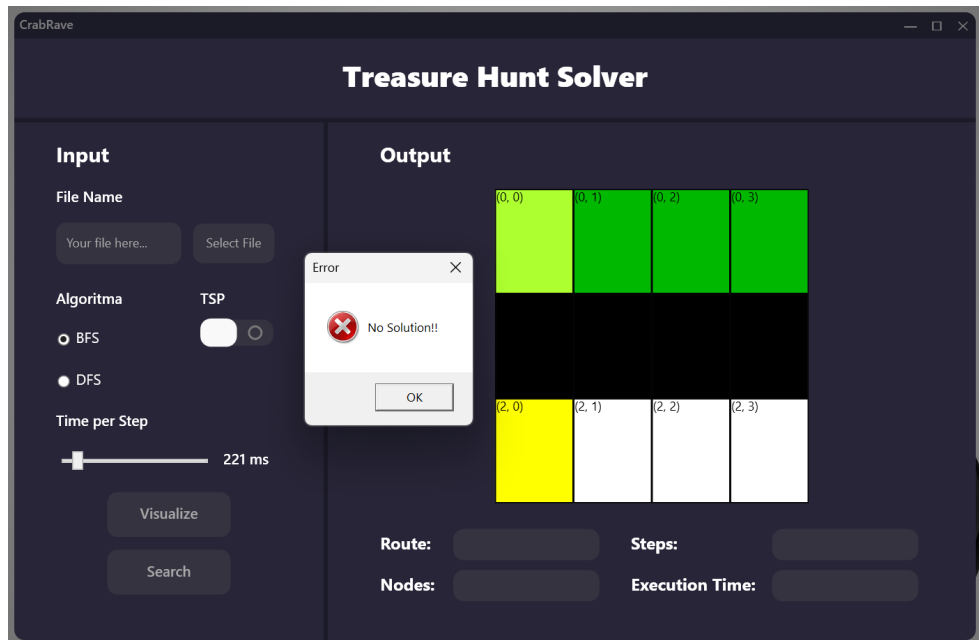
Pada kasus pencarian kali ini, kami menggunakan matrix yang berperilaku seperti graf. Kami menggunakan matrix dengan elemen simpul(*node*) yang telah dipindai di awal program dan memetakannya dengan posisi (x, y). Berikut adalah beberapa elemennya.

Tabel 3.2.1 Elemen pada BFS/DFS

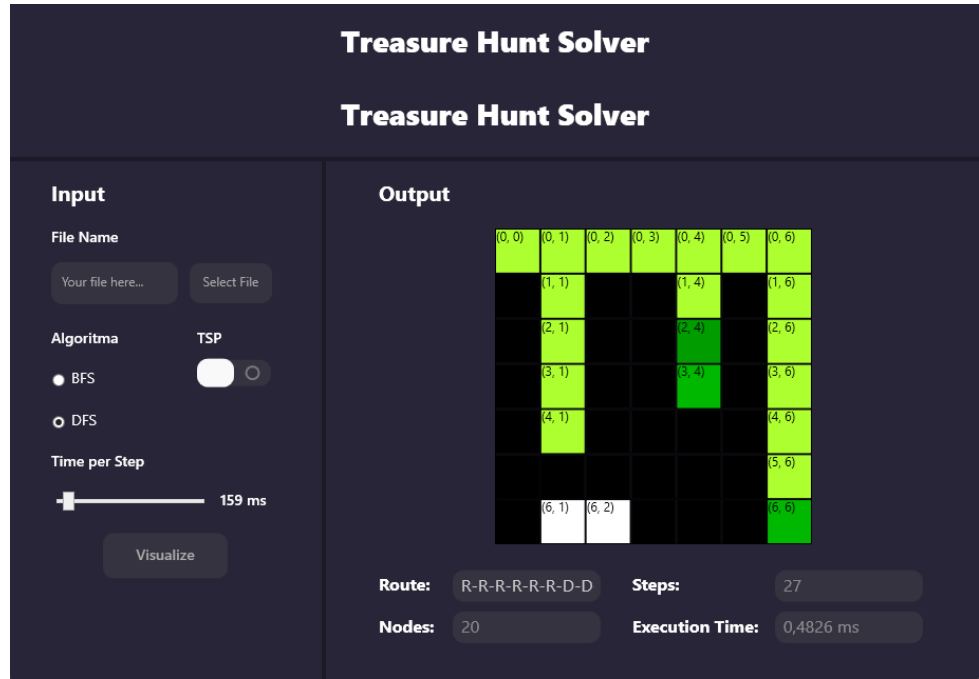
Nodes	Tiap kotak/ <i>grid</i> pada maze
Edges	Sisi dari kotak/ <i>grid</i> pada maze yang bersebelahan
Goal State	Semua <i>treasure</i> telah ditemukan, dan kembali ke <i>start</i> (jika

	TSP)
--	------

3.3. Contoh Ilustrasi Kasus Lain



Gambar 3.3.1 Tampilan aplikasi saat tidak ada solusi (treasure not accessible)



Gambar 3.3.2 Tampilan aplikasi saat ada jalur yang tidak bisa diakses

Bab 4

Implementasi dan Pengujian

4.1. Implementasi Program (Pseudocode Program Utama)

4.2.1. Implementasi BFS

```
public class BFS
    /* Variables needed in the process */
    private Queue<Node>? visitQueue // visit queue 0> store next node to
be visited in queue
    private Node? start // start node 0> store starting node
    private int treasureFound // number of treasure found (current)
    private int expectedTreasure // number of treasure expected
    private Map map // map to be searched
    private bool backtrackOn // option to use backtrack. if set off,
travelling between node can be done directly (without backtrack)

    // Result
    private ObservableCollection<Node> path // 0> store path of node
visited through the search (can include backtrack)
    private List<Node> treasureNodes // 0> store nodes that are treasure
    private int nodeVisited // 0> number of node visited
    private string route // 0> solution route (in string) after the search
    private int steps // 0> steps taken in route

    // Getter result
    public int NodeVisited { get => nodeVisited }
    public string Route { get => route }
    public int Steps { get => steps }

    public bool isRunning

    /* backtracking attributes */
    private Dictionary<Node, int>? depthOf // depth of node
    private Dictionary<Node, Node>? parentOf // 0> track the parent of
each node
    private Dictionary<Node, List<Node>>? pathToNode // 0> store path from
start to each node

    /// <summary>
    /// Default constructor, the backtrack is off
    /// </summary>
    /// <param name="-m">Map that will be searched</param>
```

```

public BFS(Map m) //ctor
    map <- m
    backtrackOn <- true
    isRunning <- false
    resetBFS()

    /// <summary>
    /// Set backtrack option to false, BFS search will not perform
backtrack
    /// </summary>
public void SetBacktrackOff()
    backtrackOn <- false
    resetBFS()

    /// <summary>
    /// Set backtrack option to true, BFS search will perform backtrack
    /// </summary>
public void SetBacktrackOn()
    backtrackOn <- true
    resetBFS()

    /// <summary>
    /// Set map to be searched.
    /// </summary>
public void SetMap(Map m)
    this.map <- m

    /// <summary>
    /// Search the treasure in the map, the result won't be returned
immediately
    /// </summary>
    /// <param name="awaitTime">Time to wait before changing node</param>
    /// <param name="tspOn">Option to use TSP. Will include go back to
starting point</param>
    /// <remarks>
    /// To access the result, you can use getter NodeVisited, Route, and
Steps.
    /// Alternatively, you can track the map since the algorithm alter it
directly.
    /// </remarks>
public async Task Search(int awaitTime, bool tspOn)
    isRunning <- true
    /* initialize starting point */
    (int xStart, int yStart) <- map.getStart()
    start <- map.map[xStart, yStart]
    start.setVisiting()

```



```

    await Task.Delay(awaitTime)

    if (backtrackOn) then //only initialize if backtrack on
        depthOf[start] <- 0
        pathToNode[start] <- new List<Node>()
    /* end of initialization */

    /* Main loop */
    Node current <- await next(start, 0)

    // next until found
    while (treasureFound != expectedTreasure and visitQueue.Count >=
0)

        //current <- next(current).Result
        current <- await next(current, awaitTime)
        if (current = null) break
    /* End of main loop */

    if (treasureFound = expectedTreasure) then // search done, go back
to the start
        if (backtrackOn and tspOn) then // only do if backtrackOn and
tspOn

            Node prev <- parentOf[current]
            while (prev != start) do
                path.Add(prev)
                await progressToNode(prev, awaitTime)
                prev <- parentOf[prev]
                path.Add(prev)
                await progressToNode(prev, awaitTime)

            /* Converting each path to treasures (List<Node>) to string */
            /* Highlighting the solution route */
            current <- start
            steps++
            await highlightNode(current, awaitTime)
            foreach (Node treasure in treasureNodes)
                // backtrack until it is possible to go to the destined node
                while (current != start and
!pathToNode[treasure].Contains(current)) do
                    route += direction(current, parentOf[current]) + '-'
                    steps++
                    current <- parentOf[current]

                // go forward until destined node
                for (int i <- depthOf[current]; i < depthOf[treasure]; i++)
                    Node n <- pathToNode[treasure][i] // node that should be

```

```

visited first before going to the next node in the queue
        route += direction(current, n)
        if (!(treasure == treasureNodes[treasureNodes.Count-1] and
i == depthOf[treasure]-1)) then
            route += '-'
        else
            if (tspOn)
                route += '-'
            steps++
            current <- n
            await highlightNode(current, awaitTime)

    if (tspOn) then
        while (current != start) do
            route += direction(current, parentOf[current])
            if (parentOf[current] != start)
                route += '-'
            steps++
            current <- parentOf[current]
/* end of finding the solution route */

isRunning <- false

/// <summary>
/// Continue the search (will update queue, nodestatus, etc)
/// </summary>
/// <param name="current">Current node</param>
/// <param name="awaitTime">Time to wait before changing node</param>
/// <returns>Next node</returns>
private async Task<Node?> next(Node current, int awaitTime)
    path.Add(current)
    await progressToNode(current, awaitTime)
    nodeVisited++

    List<Node> adjacents <- adjacentNode(current) // Guaranteed that
it has not been visited

    if (current.isTreasure()) then
        treasureFound++
        treasureNodes.Add(current)

        // if all treasure found, get out immediately
        if (treasureFound = expectedTreasure) -> current

    foreach (Node node in adjacents)
        visitQueue.Enqueue(node)

```

```

        if (backtrackOn) then
            depthOf[node] <- depthOf[current] + 1
            parentOf[node] <- current
            pathToNode[node] <- new List<Node>(pathToNode[current])
            pathToNode[node].Add(node)

    if (backtrackOn and visitQueue.Count > 0) then
        await addPathToNextNode(current, awaitTime)
    current.setVisited()

    -> visitQueue.Count > 0 ? visitQueue.Dequeue() : null

    /// <summary>
    /// Get all adjacent node from node c that is available (check node
definition of available)
    /// </summary>
    /// <param name="c">The node</param>
    /// <returns>All node adjacent to node c that is available</returns>
    private List<Node> adjacentNode(Node c)
        List<Node> nodes <- new List<Node>()

        /* The order is : Right - Down - Left - Up */
        if (map.isRightAvailable(c.x, c.y)) then
            nodes.Add(map.map[c.x, c.y + 1])
        if (map.isDownAvailable(c.x, c.y)) then
            nodes.Add(map.map[c.x + 1, c.y])
        if (map.isLeftAvailable(c.x, c.y)) then
            nodes.Add(map.map[c.x, c.y - 1])
        if (map.isUpAvailable(c.x, c.y)) then
            nodes.Add(map.map[c.x - 1, c.y])

        -> nodes

    /// <summary>
    /// Adding path from current node to the next node in the visit queue
to the search path.
    /// Basically backtrack to the next node.
    /// </summary>
    /// <param name="current">Current Node</param>
    /// <param name="awaitTime">Time to wait before changing node</param>
    private async Task addPathToNextNode(Node current, int awaitTime)
        while (visitQueue.Peek().hasBeenVisited()) // ignore this node
            visitQueue.Dequeue();

        // if next node in queue is the same level as current or deeper 0>
require backtracking and retracking

```

```

        if (visitQueue.Count > 0 and current != start and
(depthOf[current] <= depthOf[visitQueue.Peek()])) then
            if (parentOf[visitQueue.Peek()] = current) then
                // no need to find path
                ->

            // backtracking
            Node prev <- parentOf[current]

            while (!pathToNode[visitQueue.Peek()].Contains(prev) and prev
!= start) do
                path.Add(prev)
                await progressToNode(prev, awaitTime)

                prev <- parentOf[prev]
                path.Add(prev)
                await progressToNode(prev, awaitTime)

            // go forward until just before the destined queue
            for (int i <- depthOf[prev] i < depthOf[visitQueue.Peek()] - 1
i++)
                Node n <- pathToNode[visitQueue.Peek()][i] // node that
should be visited first before going to the next node in the queue
                path.Add(n)
                await progressToNode(n, awaitTime)

    /// <summary>
    /// Set all the BFS variables to default starting point
    /// </summary>
    private void resetBFS()
    {
        expectedTreasure <- map.getTreasureCount()
        visitQueue <- new Queue<Node>()
        path <- new ObservableCollection<Node>()
        treasureNodes <- new List<Node>()
        treasureFound <- 0
        steps <- -1
        nodeVisited <- 0
        route <- ""

        if (backtrackOn) then
            depthOf <- new Dictionary<Node, int>()
            parentOf <- new Dictionary<Node, Node>()
            pathToNode <- new Dictionary<Node, List<Node>>()

        private async Task progressToNode(Node n, int awaitTime)
        {
            // visit a node

```

```

        n.setVisiting()
        await Task.Delay(awaitTime)
        n.setVisited()

        /// <summary>
        /// Highlight the node as a solution node
        /// </summary>
        /// <param name="n">Node to be highlighted</param>
        /// <param name="awaitTime">Waiting time between each node
highlighting</param>
        private async Task highlightNode(Node n, int awaitTime)
        {
            n.highlightSolution()
            await Task.Delay(awaitTime)

            /// <summary>
            /// Get direction from source node to destination node
            /// </summary>
            /// <param name="source"></param>
            /// <param name="dest"></param>
            /// <returns>(R, D, L, U) based on the direction</returns>
            private string direction(Node source, Node dest)
            {
                if (source.x == dest.x) then
                    if (source.y < dest.y) then
                        -> "R"
                    else
                        -> "L"
                else
                    if (source.x < dest.x) then
                        -> "D"
                    else
                        -> "U"
            }
        }
    endclass

```

4.2.2. Implementasi DFS

```

public class DFS
{
    public Stack<Node> path // holds the path that is used
    public int stepsTaken
    public int treasureFound
    public string route
    public int nodesVisited
    public bool isRunning

    public bool isItSection(Map map, int x, int y)

```

```

int ctr <- 0

if (map.isRightAvailable(currentX, currentY)) then
  ctr <- ctr + 1
if (map.isDownAvailable(currentX, currentY)) then
  ctr <- ctr + 1
if (map.isLeftAvailable(currentX, currentY)) then
  ctr <- ctr + 1
if (map.isUpAvailable(currentX, currentY)) then
  ctr <- ctr + 1
if (ctr > 1) then
  -> true
-> false

public bool isItInTheList(List<Node> list, int x, int y)
  for (int i <- 0; i < list.Count; i++)
    if (list.ElementAt(i).x == x && list.ElementAt(i).y == y) then
      -> true
    -> false

public async Task StartDFS(Map map, bool tsp, int awaitTime)
  // priorities: R D L U
  Stack<Node> stack <- new Stack<Node>()
  List<Node> treasureNode<- new List<Node>()
  this.path <- new Stack<Node>()
  this.stepsTaken <- 0
  this.treasureFound <- 0
  this.nodesVisited <- 0
  this.route <- ""
  bool thereIsTreasure <- false
  isRunning <- true

  // get the start point
  (int currentX, int currentY) <- map.getStart()

  map.map[currentX, currentY].setVisiting()
  nodesVisited++
  await Task.Delay(awaitTime)

  stack.Push(map.map[currentX, currentY])
  this.path.Push(map.map[currentX, currentY])

  // while stack is not empty
  while (stack.Count != 0) do
    if (treasureFound = map.getTreasureCount()) then
      // if all treasure has been found

```

```

if (!tsp) then
    // if not tsp
    break
else
    // if tsp, then find the route back to start
    if (map.adjacentToStart(currentX, currentY)) then
        nodesVisited-- // adjust saat visit krustykrab di awal
        if (map.rightIsStart(currentX, currentY)) then
            // if start is at the right side of current node
            map.map[currentX, currentY].setVisited()
            currentY++
            map.map[currentX, currentY].setVisiting()
            await Task.Delay(awaitTime)
            this.stepsTaken++
            this.path.Push(map.map[currentX, currentY])
            if (this.route != "") {
                this.route += "-"
            }

            if (!map.map[currentX, currentY].hasBeenVisited())
            {
                nodesVisited++
            }
            this.route += "R"
        } else if (map.downIsStart(currentX, currentY)) {
            // if start is at the down side of current node
            map.map[currentX, currentY].setVisited()
            currentX++
            map.map[currentX, currentY].setVisiting()
            await Task.Delay(awaitTime)
            this.stepsTaken++
            this.path.Push(map.map[currentX, currentY])
            if (this.route != "") {
                this.route += "-"
            }

            if (!map.map[currentX, currentY].hasBeenVisited())
            {
                nodesVisited++
            }

            this.route += "D"
        } else if (map.leftIsStart(currentX, currentY)) {
            // if start is at the left side of current node
            map.map[currentX, currentY].setVisited()
            currentY--

```

```

        map.map[currentX, currentY].setVisiting()
        await Task.Delay(awaitTime)
        this.stepsTaken++
        this.path.Push(map.map[currentX, currentY])
        if (this.route != "") {
            this.route += "-"
        }

        if (!map.map[currentX, currentY].hasBeenVisited())
        {
            nodesVisited++
        }

        this.route += "L"
    }
    else
    {
        // if start is at the up side of current node
        map.map[currentX, currentY].setVisited()
        currentX--
        map.map[currentX, currentY].setVisiting()
        await Task.Delay(awaitTime)
        this.stepsTaken++
        this.path.Push(map.map[currentX, currentY])
        if (this.route != "") then
            this.route += "-"
        if (!map.map[currentX, currentY].hasBeenVisited())

then
            nodesVisited++

            this.route += "U"
        break

    if (map.isRightAvailable(currentX, currentY)) then
        // if the right side of current node is available
        if (map.isRightAvailable(currentX, currentY)) then
            treasureNode.Add(map.map[currentX, currentY])
            thereIsTreasure <- false
            map.map[currentX, currentY].setVisited()
            currentY++
            map.map[currentX, currentY].setVisiting()

        await Task.Delay(awaitTime)

        if (map.map[currentX, currentY].isTreasure()) then
            treasureFound++

```



```

        this.path.Push(map.map[currentX, currentY])
        stack.Push(map.map[currentX, currentY])
        this.stepsTaken++
        if (this.route != "") then
            this.route += "-"

        if (!map.map[currentX, currentY].hasBeenVisited()) then
            nodesVisited++

        this.route += "R"
    else if (map.isDownAvailable(currentX, currentY)) then
        // if the down side of current node is available
        if (map.isRightAvailable(currentX, currentY)) then
            treasureNode.Add(map.map[currentX, currentY])
            thereIsTreasure <- false
            map.map[currentX, currentY].setVisited()
            currentX++
            map.map[currentX, currentY].setVisiting()

        await Task.Delay(awaitTime)

        if (map.map[currentX, currentY].isTreasure()) then
            treasureFound++
        this.path.Push(map.map[currentX, currentY])
        stack.Push(map.map[currentX, currentY])
        this.stepsTaken++
        if (this.route != "") then
            this.route += "-"
    }

    if (!map.map[currentX, currentY].hasBeenVisited())
    {
        nodesVisited++
    }

    this.route += "D"
}
else if (map.isLeftAvailable(currentX, currentY)) {
    // if the left side of current node is available
    if (map.isRightAvailable(currentX, currentY)) then
        treasureNode.Add(map.map[currentX, currentY])
        thereIsTreasure <- false
        map.map[currentX, currentY].setVisited()
        currentY--
        map.map[currentX, currentY].setVisiting()
}

```

```

        await Task.Delay(awaitTime)

        if (map.map[currentX, currentY].isTreasure())
        {
            treasureFound++
        }
        this.path.Push(map.map[currentX, currentY])
        stack.Push(map.map[currentX, currentY])
        this.stepsTaken++
        if (this.route != "")
        {
            this.route += "-"
        }

        if (!map.map[currentX, currentY].hasBeenVisited())
        {
            nodesVisited++
        }

        this.route += "L"
    } else if (map.isUpAvailable(currentX, currentY)) {
        // if the up side of current node is available
        if (map.isRightAvailable(currentX, currentY)) then
            treasureNode.Add(map.map[currentX, currentY])
        thereIsTreasure <- false
        map.map[currentX, currentY].setVisited()
        currentX--
        map.map[currentX, currentY].setVisiting()

        await Task.Delay(awaitTime)

        if (map.map[currentX, currentY].isTreasure())
        {
            treasureFound++
        }
        this.path.Push(map.map[currentX, currentY])
        stack.Push(map.map[currentX, currentY])
        this.stepsTaken++
        if (this.route != "") then
            this.route += "-"

        if (!map.map[currentX, currentY].hasBeenVisited()) then
            nodesVisited++

        this.route += "U"
    }
else

```

```

        // if there are no available nodes to be visited from the
current node, then backtrack
        map.map[currentX, currentY].setVisited()
        Node temp <- stack.Pop()

        // if while backtracking there is treasure
        if (temp.isTreasure() && !thereIsTreasure) then
            thereIsTreasure <- true
            currentX <- temp.x
            currentY <- temp.y
            map.map[currentX, currentY].setVisiting()
            await Task.Delay(awaitTime)

        if (temp.x = currentX && temp.y = currentY) then
            if (!thereIsTreasure && !map.map[currentX,
currentY].isTreasure()) then
                // if while backtracking there are no treasures, then
delete it from the path
                // backtrack to the node before
                if (this.path.Count > 0) then
                    this.path.Pop()
                    currentX <- path.ElementAt(0).x
                    currentY <- path.ElementAt(0).y
                else
                    currentX <- path.ElementAt(0).x
                    currentY <- path.ElementAt(0).y
                    this.path.Pop()
                map.map[currentX, currentY].setVisiting()
                await Task.Delay(awaitTime)
                // check if the current node is actually a needed node
in the path
                if (isItInTheList(treasureNode, currentX, currentY))
then
                    thereIsTreasure <- true
                    // undo the newest route
                    if (this.route.Length > 0) then
                        stepsTaken--
                        this.route <- this.route.Remove(this.route.Length
- 1)

                        if (this.route.Length >= 2) then
                            this.route <-
this.route.Remove(this.route.Length - 1)
                        else
                            if (thereIsTreasure) then
                                // if while backtracking there are treasures, then add
it to the path

```

```

        this.path.Push(temp)
        this.stepsTaken++

        if (currentX < temp.x) then
            if (this.route != "") then
                this.route += "-"
            this.route += "D"
        else if (currentX > temp.x) then
            if (this.route != "") then
                this.route += "-"
            this.route += "U"
        else if (currentY < temp.y) then
            if (this.route != "") then
                this.route += "-"
            this.route += "R"
        else
            if (this.route != "")
                this.route += "-"
            this.route += "L"

        if (!thereIsTreasure) then
            this.route <- this.route.Remove(this.route.Length - 1)
            if (this.route.Length >= 2) then
                this.route <- this.route.Remove(this.route.Length
- 1)

        currentX <- temp.x
        currentY <- temp.y
        stack.Push(map.map[currentX, currentY])
        map.map[currentX, currentY].setVisiting()
        await Task.Delay(awaitTime)

        // check if the current node is actually a needed node in
the path

        if (isItInTheList(treasureNode, currentX, currentY)) then
            thereIsTreasure <- true
    if (stack.Count = 0) then
        if (map.isRightAvailable(currentX, currentY)) then
            stack.Push(map.map[currentX, currentY])
        else if (map.isDownAvailable(currentX, currentY)) then
            stack.Push(map.map[currentX, currentY])
        else if (map.isLeftAvailable(currentX, currentY)) then
            stack.Push(map.map[currentX, currentY])
        else if (map.isUpAvailable(currentX, currentY)) then
            stack.Push(map.map[currentX, currentY])

        // highlight solution

```

```
while (this.path.Count != 0) do
  Node temp <- this.path.Pop()
  map.map[temp.x, temp.y].highlightSolution()
  await Task.Delay(awaitTime)

isRunning <- false
endclass
```

4.2. Penjelasan Struktur Data

4.2.1. Node

Node merupakan elemen terkecil dari maze. Maze memiliki 4 jenis *node* yang, yaitu *start*, *path*, *treasure*, dan halangan. *Node* juga memiliki atribut-atribut lain seperti *x* & *y* yang merupakan index *node* tersebut pada matrix, *numOfVisits* yang menunjukkan berapa kali *node* tersebut telah dikunjungi, warna dari *node* tersebut, serta *prog* yang merupakan status dari *node* tersebut. Status-status pada *prog* berupa *Unvisited*, *Visited*, dan *InVisit*.

Penjelasan warna node pada UI adalah sebagai berikut.

1. Hitam : Jalan yang tidak dapat diakses
2. Putih : Jalan yang dapat diakses
3. Cream : Krusty Krab (titik start)
4. Kuning : Treasure
5. Biru : Node yang sedang dikunjungi
6. Hijau muda : Node yang sudah dikunjungi
7. Hijau neon : Highlight solusi saat proses selesai

Semakin sering node itu dikunjungi maka warna hijau muda yang terlihat akan semakin gelap, ini agar menandakan bahwa node mana yang sering dikunjungi saat proses pencarian.

4.2.2. Map

Maze akan disimpan dalam struktur data Map. Map menyimpan struktur atau bentuk maze dalam bentuk *matrix of nodes* yang dapat diakses satu persatu dengan menggunakan *index*. Map juga menyimpan dua buah integer yang merupakan batas *index* maksimum dari maze tersebut.

Secara *default* ketika map dibuat, semua *node* yang terdapat pada matrik adalah rintangan. Sehingga *node-node* seperti *start*, *path*, dan *treasure* harus di-set satu persatu pada matrix.

4.2.3. List

List merupakan struktur data yang berupa daftar. *List* digunakan pada BFS sebagai tempat penyimpanan *node* yang merupakan *treasure* dan juga path dari *starting node* ke *node* tertentu. *List* juga digunakan pada DFS sebagai tempat penyimpanan *node* yang merupakan *node* yang diperlukan untuk mendapatkan *treasure*.

4.2.4. Queue

Queue merupakan struktur data yang berupa antrian, sehingga elemen-elemen pada *queue* tersimpan dalam aturan FIFO (*first in first out*). *Queue* digunakan pada BFS sebagai tempat penyimpanan *node* yang akan dikunjungi selanjutnya.

4.2.5. Stack

Stack merupakan struktur data yang berupa tumpukan data, sehingga elemen-elemen pada *stack* disimpan dengan aturan LIFO (*last in first out*). *Stack* digunakan pada DFS sebagai tempat penyimpanan *node-node* yang berupa lintasan penelusuran maze, serta menyimpan *node-node* yang merupakan lintasan solusi yang didapat dari proses pencarian *treasure*.

4.2.6. Dictionary

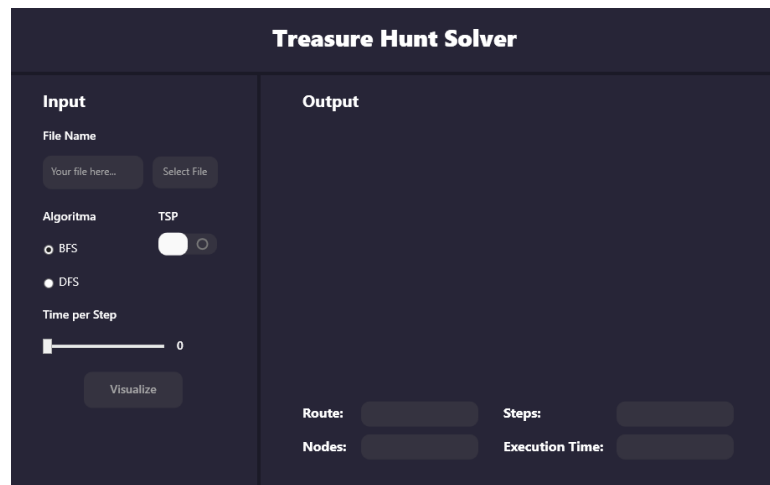
Dictionary adalah struktur data yang berisikan pasangan *key* dan *value*. *Dictionary* digunakan pada BFS untuk menyimpan informasi mengenai node seperti kedalaman node, path menuju node, dan parent dari sebuah node. Dalam hal ini, node menjadi *key*-nya.

4.3. Penjelasan Tata Cara Penggunaan Program

Untuk menggunakan program ini diperlukan beberapa *requirement* sebagai berikut.

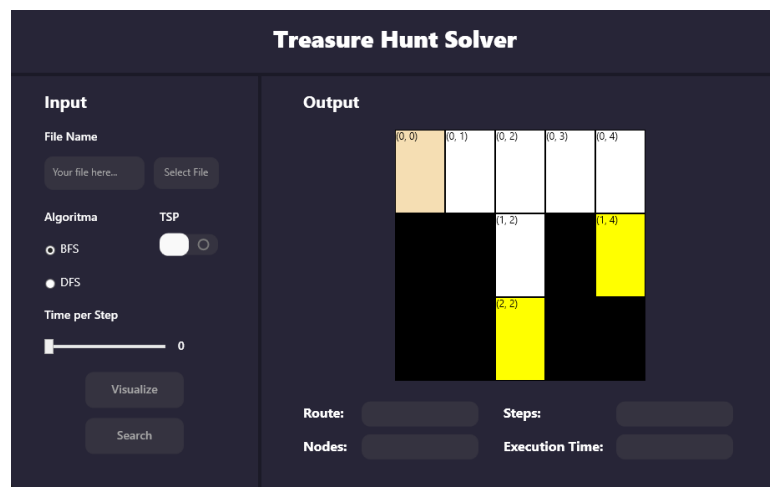
1. Memiliki .NET Framework setidaknya v4.8

2. Komputer menggunakan OS Windows



Gambar 4.3.1 Interface awal program

Pada saat memulai program, pengguna diminta untuk memasukkan file txt yang berupa *layout* maze yang ingin dijelajahi. Pengguna dapat memasukkan file *layout* tersebut dengan menekan tombol *Select File*. Setelah menekan tombol tersebut, nanti program akan menampilkan *file explorer* yang dapat digunakan pengguna untuk memilih file mana yang ingin dimasukkan. Setelah pengguna memasukkan file tersebut, pengguna kemudian dapat menekan tombol *Visualize* untuk memvisualisasikn mazenya.

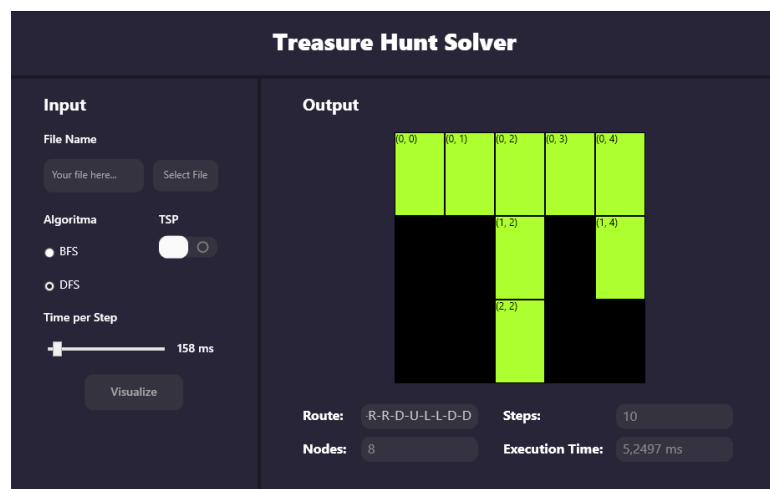


Gambar 4.3.2 Interface awal program setelah menekan Visualize

Setelah maze sukses divisualisasi oleh program, pengguna kemudian dapat memilih algoritma pencarian apa yang ingin digunakan, serta apakah pengguna ingin

algoritma tersebut mencari jalan pulang ke *start* atau tidak dengan meng-*toggle* opsi TSP. Pengguna juga bisa mengatur kecepatan visualisasi langkah pencarian dengan menggeser *scroll bar Time per Step*. Setelah algoritma pencarian dipilih, pengguna dapat menekan tombol *Search* untuk mencari rutenya.

Program kemudian akan meng-*highlight* jalur solusi yang didapat dari pencarian pada visualisasi maze. Hasil-hasil lainnya dari pencarian algoritma tersebut kemudian ditampilkan pada bagian bawah kanan *interface*. Hasil-hasil tersebut berupa rute, *node*, jumlah langkah, serta waktu eksekusi program.



Gambar 4.3.3 Interface saat menampilkan output

Ketika program telah menampilkan hasilnya, pengguna kemudian dapat mengulang langkah-langkah diatas jika ingin menyimulasikan maze lainnya.

4.4. Hasil Pengujian

4.4.1. Test Case 1

Tabel 4.4.1.1 Hasil dari test case 1

Algoritma	Hasil
-----------	-------

BFS

Treasure Hunt Solver

Input

File Name

Algoritma
☐ BFS ☒ DFS

Time per Step
 129 ms

Output

Route: R-U-U-D-R-R-D Steps: 7

Nodes: 11 Execution Time: 0,7921 ms

DFS

Treasure Hunt Solver

Input

File Name

Algoritma
☐ BFS ☒ DFS

Time per Step
 129 ms

Output

Route: R-D-D-R-R-U-U-L Steps: 11

Nodes: 12 Execution Time: 0,3549 ms

TSP dengan BFS	<div> <div>Treasure Hunt Solver</div> <div> <div>Input</div> <div> File Name <div>Your file here... Select File</div> </div> <div> Algoritma <div> <input checked="" type="radio"/> BFS <input type="radio"/> DFS </div> </div> <div> Time per Step <div> <div></div> 101 ms </div> </div> <div>Visualize</div> </div> <div> <div>Output</div> <div> </div> <div> Route: R-U-U-D-R-R-D-L Steps: 12 Nodes: 11 Execution Time: 0,5288 ms </div> </div> </div>
TSP dengan DFS	<div> <div>Treasure Hunt Solver</div> <div> <div>Input</div> <div> File Name <div>Your file here... Select File</div> </div> <div> Algoritma <div> <input type="radio"/> BFS <input checked="" type="radio"/> DFS </div> </div> <div> Time per Step <div> <div></div> 129 ms </div> </div> <div>Visualize</div> </div> <div> <div>Output</div> <div> </div> <div> Route: R-D-D-R-R-U-U-L Steps: 22 Nodes: 12 Execution Time: 0,4431 ms </div> </div> </div>

4.4.2. Test Case 2

Tabel 4.4.2.1 Hasil dari test case 2

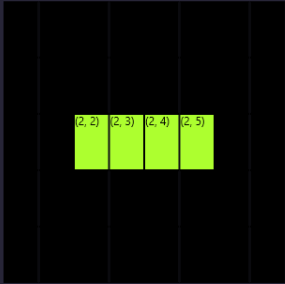
Algoritma	Hasil
-----------	-------

BFS

Treasure Hunt Solver

Input
File Name

Algoritma
☐ BFS ☒ DFS
Time per Step
 129 ms

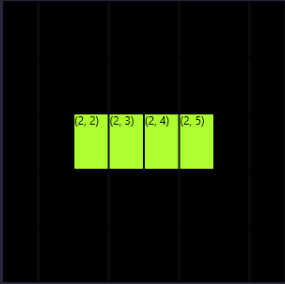
Output

Route: Steps:
Nodes: Execution Time:

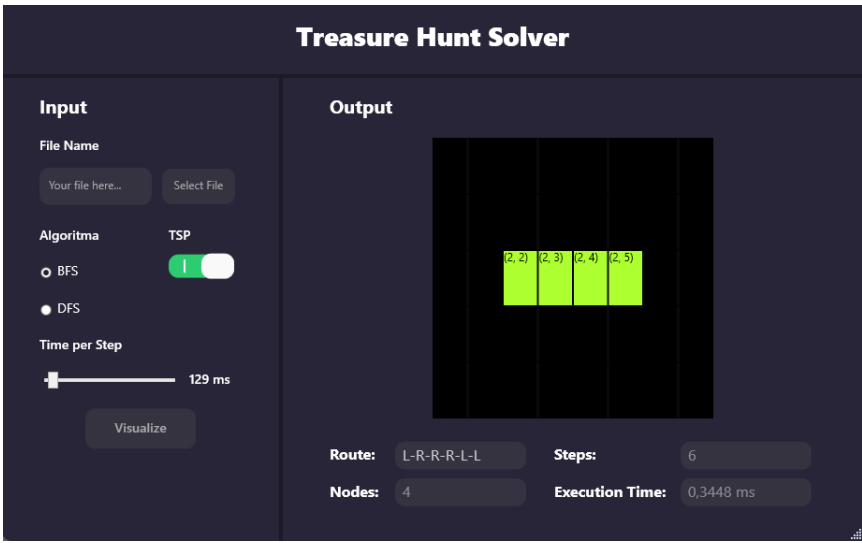
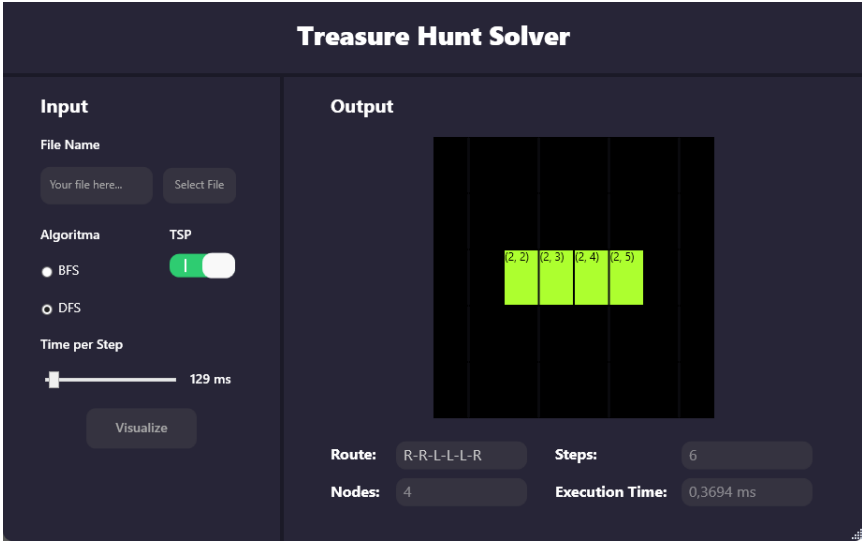
DFS

Treasure Hunt Solver

Input
File Name

Algoritma
☒ BFS ☐ DFS
Time per Step
 129 ms

Output

Route: Steps:
Nodes: Execution Time:

TSP dengan BFS	 <p>Treasure Hunt Solver</p> <p>Input</p> <p>File Name Your file here... Select File</p> <p>Algoritma <input checked="" type="radio"/> BFS <input type="radio"/> DFS </p> <p>Time per Step <input type="range"/> 129 ms </p> <p>Visualize</p> <p>Output</p> <p>Route: L-R-R-R-L-L Steps: 6</p> <p>Nodes: 4 Execution Time: 0,3448 ms</p>
TSP dengan DFS	 <p>Treasure Hunt Solver</p> <p>Input</p> <p>File Name Your file here... Select File</p> <p>Algoritma <input type="radio"/> BFS <input checked="" type="radio"/> DFS </p> <p>Time per Step <input type="range"/> 129 ms </p> <p>Visualize</p> <p>Output</p> <p>Route: R-R-L-L-L-R Steps: 6</p> <p>Nodes: 4 Execution Time: 0,3694 ms</p>

4.4.3. Test Case 3

Tabel 4.4.3.1 Hasil dari test case 3

Algoritma	Hasil
-----------	-------

BFS

Treasure Hunt Solver

Input
File Name
sampel-3.txt Select File
Algoritma
☐ BFS ☒ TSP
☐ DFS
Time per Step
 0
Visualize

Output

Error
The character in txt file is out of format
OK

Route: Steps:
Nodes: Execution Time:

DFS

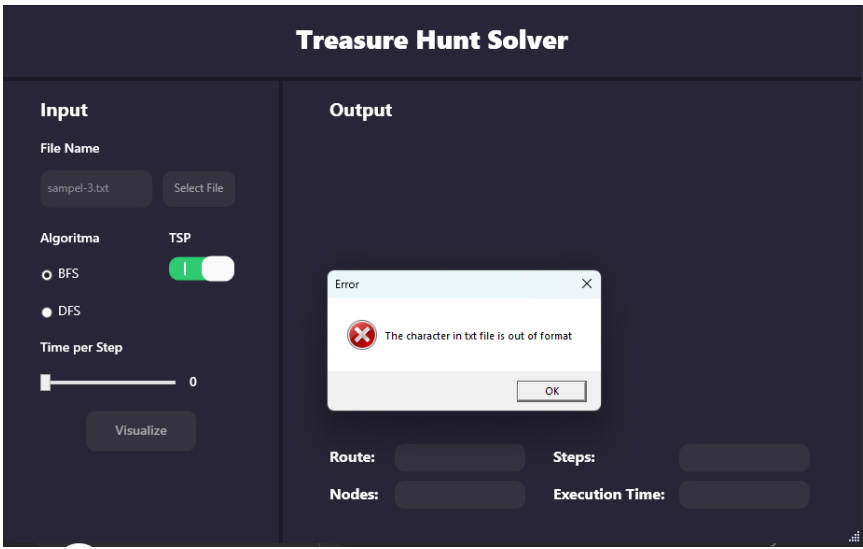
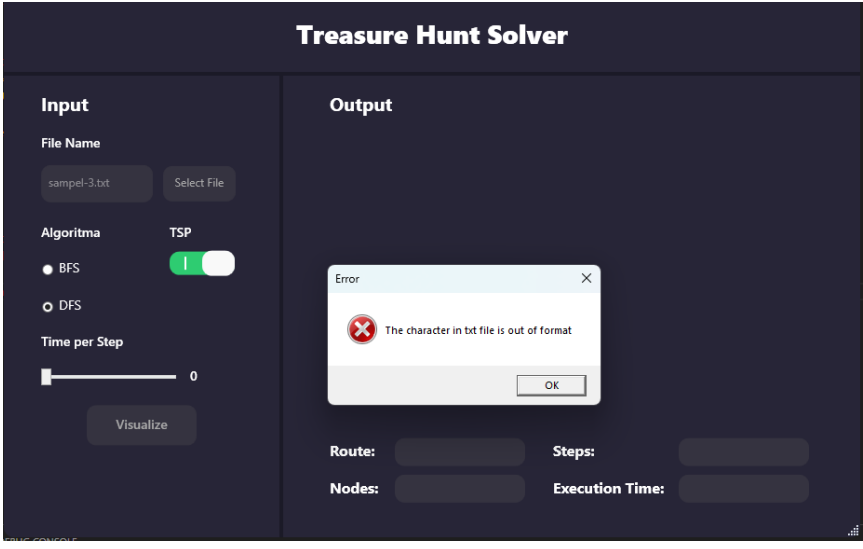
Treasure Hunt Solver

Input
File Name
sampel-3.txt Select File
Algoritma
☒ BFS ☐ TSP
☐ DFS
Time per Step
 0
Visualize

Output

Error
The character in txt file is out of format
OK

Route: Steps:
Nodes: Execution Time:

TSP dengan BFS	 <p>The screenshot shows the 'Treasure Hunt Solver' application. In the 'Input' section, 'File Name' is 'sampel-3.txt' and 'Algoritma' is set to 'TSP' (indicated by a green toggle). An error dialog box is displayed in the center, stating 'Error' and 'The character in txt file is out of format'. The 'Output' section shows fields for 'Route:', 'Steps:', 'Nodes:', and 'Execution Time:', all of which are currently empty.</p>
TSP dengan DFS	 <p>This screenshot is identical to the one above, showing the 'Treasure Hunt Solver' interface with the same error message and settings. The 'Algoritma' is still 'TSP' and the error dialog is present.</p>

4.4.4. Test Case 4

Tabel 4.4.4.1 Hasil dari test case 4

Algoritma	Hasil
-----------	-------

BFS

Treasure Hunt Solver

Input

File Name

Algoritma **TSP**

☐ BFS ☒ DFS

Time per Step
 129 ms

Output

Route: D-D-D-D-D-R-R-F Steps: 10

Nodes: 21 Execution Time: 2,9712 ms

DFS

Treasure Hunt Solver

Input

File Name

Algoritma **TSP**

☒ BFS ☐ DFS

Time per Step
 129 ms

Output

Route: D-R-D-R-D-R-D-R Steps: 10

Nodes: 11 Execution Time: 0,5536 ms

TSP dengan BFS	<div> <div> <h3>Treasure Hunt Solver</h3> <div> <div>Input</div> <div> File Name <div> Your file here... Select File </div> </div> <div> Algoritma <div> <div>TSP</div> <div> <input checked="" type="radio"/> BFS <input type="radio"/> DFS </div> </div> <div> Time per Step <div> <div></div> <div>129 ms</div> </div> </div> <div>Visualize</div> </div> </div> <div> <div>Output</div> <div> <div> Route: D-D-D-D-D-R-R-F Steps: 20 </div> <div> Nodes: 21 Execution Time: 0,4463 ms </div> </div> </div> </div> </div>
TSP dengan DFS	<div> <div> <h3>Treasure Hunt Solver</h3> <div> <div>Input</div> <div> File Name <div> Your file here... Select File </div> </div> <div> Algoritma <div> <div> <input type="radio"/> BFS <input checked="" type="radio"/> DFS </div> </div> <div> Time per Step <div> <div></div> <div>156 ms</div> </div> </div> <div>Visualize</div> </div> </div> <div> <div>Output</div> <div> <div> Route: D-R-D-R-D-R-D-R Steps: 20 </div> <div> Nodes: 21 Execution Time: 0,4862 ms </div> </div> </div> </div> </div>

4.4.5. Test Case 5

Tabel 4.4.5.1 Hasil dari test case 5

Algoritma	Hasil
-----------	-------

BFS

Treasure Hunt Solver

Input

File Name

Algoritma **TSP**

☐ BFS ☒ DFS

Time per Step
 101 ms

Output

(0, 0)	(0, 1)	(0, 2)	
(1, 1)	(1, 2)		
(2, 1)	(2, 2)		
(3, 1)	(3, 2)		
(4, 1)	(4, 2)		
(5, 1)	(5, 2)		
(6, 1)	(6, 2)		
(7, 1)	(7, 2)		
(8, 1)	(8, 2)		
(9, 1)	(9, 2)		
(10, 1)	(10, 2)		

Route: R-D-D-D-D-D-I Steps: 11

Nodes: 22 Execution Time: 0,2911 ms

DFS

Treasure Hunt Solver

Input

File Name

Algoritma **TSP**

☒ BFS ☐ DFS

Time per Step
 101 ms

Output

(0, 0)	(0, 1)	(0, 2)	
(1, 1)	(1, 2)		
(2, 1)	(2, 2)		
(3, 1)	(3, 2)		
(4, 1)	(4, 2)		
(5, 1)	(5, 2)		
(6, 1)	(6, 2)		
(7, 1)	(7, 2)		
(8, 1)	(8, 2)		
(9, 1)	(9, 2)		
(10, 1)	(10, 2)		

Route: R-R-D-D-D-D-I Steps: 22

Nodes: 23 Execution Time: 0,2894 ms

TSP dengan BFS	 <p>Treasure Hunt Solver</p> <p>Input</p> <p>File Name Your file here... Select File</p> <p>Algoritma TSP <input checked="" type="radio"/> BFS <input type="radio"/> DFS</p> <p>Time per Step 101 ms</p> <p>Visualize</p> <p>Output</p> <p>Route: D-D-D-D-U-U-U Steps: 22</p> <p>Nodes: 22 Execution Time: 0,2577 ms</p>
TSP dengan DFS	 <p>Treasure Hunt Solver</p> <p>Input</p> <p>File Name Your file here... Select File</p> <p>Algoritma TSP <input type="radio"/> BFS <input checked="" type="radio"/> DFS</p> <p>Time per Step 101 ms</p> <p>Visualize</p> <p>Output</p> <p>Route: D-D-D-D-L-U-L Steps: 44</p> <p>Nodes: 23 Execution Time: 0,2513 ms</p>

4.5. Analisis Desain Solusi Algoritma BFS dan DFS

Dari hasil pengujian yang sudah dilakukan, algoritma DFS dan BFS sudah menjalankan tugasnya dengan baik dan memberikan hasil yang sesuai ketika diminta mencari semua *treasure* yang ada pada *maze*. Algoritma DFS dan BFS yang kami lakukan juga berjalan dengan cukup cepat. Algoritma DFS memiliki kompleksitas waktu $O(b^m)$ dimana b adalah percabangan maksimum (dalam hal ini 4) dan m adalah kedalaman maksimum dari ruang status. Sementara itu, algoritma BFS memiliki kompleksitas $O(b^d)$ dimana b adalah percabangan maksimum (juga 4) dan d adalah kedalaman dari simpul tujuan.

Untuk memenuhi bonus, kami membuat agar bisa dilakukan dengan TSP. Langkah-langkah yang diambil memang tidak optimal, tetapi kami memakai teknik heuristik agar jalan yang dilalui saat balik lagi ke titik awal adalah yang optimal. Langkah-langkah pencarian juga dapat dilakukan dengan jeda yang berbeda-beda agar setiap langkahnya dapat terbaca dengan jelas. Rata-rata *execution time* pada program kami adalah dalam rentang 0 - 5 ms, tergantung seberapa besar peta dan algoritma apa yang dipakai.

Bab 5

Kesimpulan dan Saran

5.1. Kesimpulan

Dari tugas ini, kami berhasil membuat Aplikasi Desktop yang dapat melakukan pencarian *treasure* dari sebuah peta (*maze*) metode DFS dan BFS. Pencarian ini digunakan untuk mencari seluruh *treasure* yang sudah ditentukan, serta dapat memilih opsi untuk melakukannya dengan TSP (akan balik lagi ke tempat semula) atau tidak.

Dari pengerjaan tugas besar ini, kami mendapatkan beberapa kesimpulan, yaitu sebagai berikut.

1. Algoritma BFS memakan lebih banyak memori karena menyimpan semua semua node pada graf sedangkan DFS tidak melakukan penyimpanan memori.
2. Jika *treasure* pada *maze* berada di jalur yang sama dan *maze* tidak bercabang-cabang, waktu yang diperlukan algoritma DFS relatif lebih cepat dibandingkan BFS dalam mencari *treasure*. Hal ini terjadi karena DFS tidak mencari setiap level, tetapi lanjut hingga tidak ada node lagi. Kekurangannya adalah DFS bisa saja mencari terus-menerus ke dalam jalur *maze* namun tidak mendapatkan *treasure* apa-apa. (*hit or miss*)
3. Namun, jika *maze* bercabang dan *treasure* tersebar di cabang-cabang tersebut, maka waktu yang diperlukan algoritma BFS akan relatif lebih cepat dibandingkan DFS. Hal ini disebabkan karena algoritma BFS mengiterasi tiap level n terlebih dahulu sehingga tidak perlu mencari hingga jalur pada *maze* buntu, lalu *backtracking* untuk mencari solusinya. (*slowly, but surely*)
4. Secara umum, algoritma BFS menghasilkan jalur yang lebih optimal dibandingkan DFS. Hal ini disebabkan karena BFS selalu mencari jalur pada setiap levelnya, sehingga jalur yang didapatkan selalu memiliki kedalaman yang paling rendah.

5.2. Saran

Untuk spesifikasi program, sedikit melenceng dari topik BFS dan DFS, karena diminta untuk adanya proses *backtracking* pada BFS. Ini dilakukan agar visualisasinya

tidak lompat-lompat. Menurut kami, hal tersebut menyebabkan banyaknya kekeliruan dalam pengerjaan karena persoalan yang dibuat menjadi tidak murni BFS dan DFS, tetapi ada unsur *backtracking* pada BFS-nya.

Selain itu, spesifikasi bonus untuk Travelling Salesman Problem (TSP) kurang sesuai dengan definisi TSP pada umumnya, dimana definisi persoalan TSP adalah mengunjungi setiap simpul satu kali dan kembali ke simpul awal. Akan tetapi, pada spesifikasi tugas ini justru diperbolehkan melewati simpul lebih dari satu kali, sehingga sangat melenceng dari definisi TSP. Selain itu, BFS dan DFS secara umum bukanlah algoritma yang cocok untuk menyelesaikan permasalahan TSP.

Algoritma BFS sangatlah cocok jika kami diharuskan untuk menemukan solusi tertentu karena algoritma tersebut akan menjelajahi graf secara merata, jadi jika terdapat permasalahan baru seharusnya BFS dengan beberapa modifikasi seperti penambahan parameter dapat menyelesaikan permasalahan tersebut.

Selain itu, aplikasi yang kami gunakan ini hanya dapat dijalankan di OS Windows, tidak *multi-platform*, sehingga tidak semua orang dapat mencobanya. Untuk kedepannya, lebih baik menggunakan *framework* yang *multi-platform* sehingga semua orang dapat mencobanya.

5.3. Refleksi

Sebelum melakukan pembuatan kode, sebaiknya dilakukan perancangan struktur yang lebih matang, sehingga tidak ada perbedaan maupun kesalahpahaman dalam pembuatan program. Selain itu, fitur pada *version control* yang digunakan (dalam hal ini github) dapat dimanfaatkan dengan lebih maksimal, seperti penggunaan *.gitignore* dan juga branching yang dapat mengurangi kode yang konflik.

5.4. Tanggapan

Dalam pelaksanaan tubes kali ini, banyak sekali *challenge* yang didapatkan. Salah satunya adalah belajar bahasa baru C# dengan langsung membuat aplikasi dengan GUI. Ini adalah *challenge* terberat karena untuk membuat GUI, harus belajar dahulu dan banyak melakukan eksplorasi. Selain itu, proses untuk sinkronisasi GUI dan aplikasi pun cukup memakan waktu yang lama. Namun, dengan adanya tubes ini, kami menjadi bertambah ilmu dan juga pengalaman yang tidak akan terlupakan.

Link Penting

Link github : https://github.com/bangkitdc/Tubes2_CrabRave.git
Link video demo : https://youtu.be/JSG_nvRen98

Daftar Pustaka

edunex.itb.ac.id (Diakses pada tanggal 18 Maret 2023)

Munir, Rinaldi. 2021. “Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1)”.
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>,
diakses 18 Maret 2023.

Munir, Rinaldi. 2021. “Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1)”.
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>,
diakses 18 Maret 2023.

[https://id.wikipedia.org/wiki/C_Sharp_\(bahasa_pemrograman\)#:~:text=Tujuan%20desain,-Standar%20European%20Computer&text=Bahasa%20pemrograman%20C%23%20ditujukan%20untuk.pemrograman%20C%20dan%20C%2B%2B](https://id.wikipedia.org/wiki/C_Sharp_(bahasa_pemrograman)#:~:text=Tujuan%20desain,-Standar%20European%20Computer&text=Bahasa%20pemrograman%20C%23%20ditujukan%20untuk.pemrograman%20C%20dan%20C%2B%2B).

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

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

Kampa Plays, C# WPF Tutorial Youtube Playlist.

https://www.youtube.com/watch?v=t9ivUosw_iI&list=PLih2KERbY1HHOOJ2C6FOrVXIwg4AZ-hk1