

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

*Diajukan sebagai pemenuhan Tugas Besar II.*



Oleh:

Kelompok 37 (*okebebasya*)

1. 13521067 - Yobel Dean Christopher
2. 13521137 - Michael Utama
3. 13521173 - Dewana Gustavus

Dosen Pengampu : Dr. Ir. Rinaldi Munir, MT.

IF2211 - Strategi Algoritma

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

**2023**

## DAFTAR ISI

<b>BAB I.....</b>	<b>2</b>
<b>BAB II.....</b>	<b>4</b>
2.1 Graph Traversal.....	4
2.2 Bread-First Search (BFS).....	4
2.3 Depth-First Search (DFS).....	5
2.4 C# Desktop Application Development.....	6
<b>BAB III.....</b>	<b>8</b>
3.1 Langkah-Langkah Pemecahan Masalah.....	8
3.2 Mapping Persoalan Menjadi Elemen-Elemen Algoritma BFS dan DFS.....	9
3.3 Ilustrasi Kasus Lain.....	10
<b>BAB IV.....</b>	<b>11</b>
4.1 Implementasi Program.....	11
4.1.1 BFS penulis sudah menyerah menulis notasi algoritmik.....	11
4.1.2 DFS.....	12
4.2 Penjelasan Struktur Data.....	13
4.3 Penjelasan Tata Cara Penggunaan Program.....	13
4.3.1 Prerequisite.....	13
4.3.2 Build dan Run.....	13
4.3.3 User Interface Program.....	14
4.4 Hasil Pengujian.....	15
4.5 Analisis Desain Algoritma.....	18
<b>BAB V.....</b>	<b>20</b>
<b>Daftar Pustaka.....</b>	<b>21</b>
<b>Lampiran.....</b>	<b>22</b>



## **BAB I**

### **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

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 II**

### **LANDASAN TEORI**

#### **2.1 Graph Traversal**

Graph traversal adalah salah satu konsep dasar dalam pemrograman yang digunakan untuk menjelajahi atau melintasi setiap simpul atau node dalam sebuah graf. Graf adalah struktur data yang terdiri dari simpul dan edge yang menghubungkan simpul-simpul tersebut. Dalam beberapa aplikasi, seperti analisis jaringan, pemodelan sosial, dan pengolahan citra, graph traversal sangat penting untuk menemukan informasi yang berharga dan pola dalam data.

Graph traversal dapat diterapkan dalam berbagai macam jenis graf, termasuk graf berarah, graf tak berarah, graf terhubung, dan graf tak terhubung. Dalam graph traversal, terdapat dua jenis metode yang umum digunakan, yaitu breadth-first search (BFS) dan depth-first search (DFS). Kedua metode ini memiliki kelebihan dan kekurangan masing-masing, tergantung pada jenis graf yang diberikan dan tujuan dari aplikasi yang akan digunakan. Pada dasarnya, graph traversal merupakan fondasi dasar dalam pemrograman yang harus dikuasai oleh setiap programmer untuk memahami berbagai konsep dan algoritma yang lebih kompleks.

#### **2.2 Breadth-First Search (BFS)**

Breadth-First Search (BFS) adalah salah satu metode dalam graph traversal yang digunakan untuk mengunjungi semua simpul atau node dalam sebuah graf. Pada dasarnya, BFS mengunjungi semua simpul pada level yang sama sebelum bergerak ke level selanjutnya. Algoritma BFS dimulai dari simpul awal dan secara berurutan menelusuri semua simpul yang terhubung dengan simpul tersebut pada level yang sama, kemudian bergerak ke level selanjutnya. Algoritma BFS sangat efektif dalam menemukan jalur terpendek dalam graf tak berbobot, serta memecahkan masalah terkait jaringan dan rute.

Untuk mengimplementasikan algoritma BFS, kita dapat menggunakan struktur data queue untuk mengurutkan antrian simpul yang akan dijelajahi. Pertama-tama, simpul awal dimasukkan ke dalam queue. Selanjutnya, simpul tersebut dikeluarkan dari queue dan semua simpul tetangga yang terhubung langsung dengan simpul tersebut dimasukkan ke dalam queue. Hal ini dilakukan hingga semua simpul telah dijelajahi. Selain itu, kita perlu mengatur daftar simpul yang telah dikunjungi untuk menghindari pengulangan. Algoritma BFS membutuhkan



waktu  $O(V+E)$  untuk mengeksplorasi seluruh simpul dan edge dalam graf, dengan  $V$  adalah jumlah simpul dan  $E$  adalah jumlah edge.

BFS digunakan dalam banyak aplikasi seperti perutean jaringan, pencarian jalur terpendek dalam graf, dan pengaturan saluran di dalam pemrosesan gambar dan video. Algoritma BFS juga dapat diadaptasi untuk digunakan dalam pemecahan masalah terkait jaringan, seperti pemodelan jaringan sosial dan analisis keamanan jaringan. Algoritma BFS juga dapat diterapkan dalam game dan animasi, seperti dalam game puzzle atau dalam membuat efek animasi pada gerakan karakter.

### 2.3 Depth-First Search (DFS)

Depth-First Search (DFS) adalah metode graph traversal yang digunakan untuk mengunjungi semua simpul atau node dalam sebuah graf. DFS menggunakan konsep stack atau rekursi untuk mengunjungi simpul secara berurutan dan menelusuri semua simpul yang terhubung dengan simpul tersebut secara mendalam. Pada dasarnya, DFS melihat sejauh mungkin dari simpul awal sebelum kembali dan mengunjungi simpul lainnya.

Implementasi algoritma DFS dapat dilakukan dengan dua cara, yaitu dengan menggunakan rekursi atau dengan menggunakan stack. Pada implementasi dengan rekursi, algoritma akan memanggil dirinya sendiri untuk menelusuri simpul-simpul yang terhubung, sedangkan pada implementasi dengan stack, simpul-simpul yang terhubung akan dimasukkan ke dalam stack dan dikunjungi secara berurutan hingga stack kosong. Waktu eksekusi algoritma DFS adalah  $O(V+E)$ , dengan  $V$  adalah jumlah simpul dan  $E$  adalah jumlah edge.

DFS sering digunakan dalam pencarian jalur pada graf, pencarian pattern dalam sebuah string atau teks, dan analisis jaringan. DFS juga dapat digunakan untuk menyelesaikan masalah dalam kecerdasan buatan seperti pengolahan bahasa alami, pembelajaran mesin, dan pengenalan pola. DFS juga dapat diterapkan dalam pemecahan masalah optimisasi, seperti dalam perencanaan produksi dan pemecahan masalah routing dalam jaringan.

## **2.4 C# Desktop Application Development**

C# desktop application development adalah proses pembuatan aplikasi desktop menggunakan bahasa pemrograman C#. C# merupakan bahasa pemrograman yang dikembangkan oleh Microsoft dan digunakan untuk mengembangkan aplikasi desktop, web, dan mobile. Aplikasi desktop adalah aplikasi yang dirancang untuk diinstal dan dijalankan di komputer pengguna, berbeda dengan aplikasi web yang diakses melalui browser.

Dalam proses pembuatan aplikasi desktop menggunakan C#, pengembang dapat menggunakan berbagai jenis tools dan framework yang disediakan oleh Microsoft, seperti .NET Framework, Windows Forms, dan WPF (Windows Presentation Foundation). .NET Framework adalah kerangka kerja untuk membangun aplikasi dengan C# dan menyediakan banyak fitur dan fungsi yang dapat digunakan oleh pengembang. Windows Forms adalah library yang menyediakan antarmuka pengguna berbasis formulir, sedangkan WPF adalah framework yang menyediakan fitur untuk membuat aplikasi dengan antarmuka pengguna yang lebih interaktif dan modern.

Pengembangan aplikasi desktop dengan C# melibatkan proses desain dan implementasi antarmuka pengguna, pemrograman logika bisnis aplikasi, dan integrasi dengan database dan sumber daya lainnya. Aplikasi desktop yang dibuat dengan C# dapat diinstal di sistem operasi Windows dan dapat digunakan untuk berbagai tujuan, seperti aplikasi bisnis, aplikasi pengolahan data, aplikasi multimedia, aplikasi permainan, dan banyak lagi. Pengembangan aplikasi desktop dengan C# juga dapat dilakukan dengan menggunakan IDE seperti Visual Studio, yang menyediakan banyak fitur dan alat bantu untuk memudahkan proses pengembangan.

Berikut adalah langkah-langkah untuk membuat aplikasi desktop menggunakan Windows Form App:

1. Buat project baru dengan membuka aplikasi Visual Studio 2022 dan memilih "Create a new project". Pilih "Windows Forms App (.NET Framework)" pada jendela "Create a new project". Isi nama project pada isian "Project name" dan tekan tombol "Create".
2. Tambahkan komponen-komponen pada aplikasi desktop dengan memilih menu Toolbox untuk membuka jendela Toolbox fly-out. Untuk memodifikasi komponen, dapat mengubah properti pada jendela properties.

3. Tambahkan kode pada form dengan membuka jendela Form1.cs [Design] dan menulis kode pada file Form1.cs.
4. Jalankan aplikasi dengan menekan tombol Start pada menu.



## **BAB III**

### **ANALISIS PEMECAHAN MASALAH**

#### **3.1 Langkah-Langkah Pemecahan Masalah**

Pertama-tama, langkah awal adalah dengan membagi permasalahan utama menjadi beberapa permasalahan kecil sehingga dapat mempermudah pencarian solusi. Dalam hal ini, permasalahan utama adalah pencarian rute untuk memperoleh seluruh treasure atau harta karun yang ada. Permasalahan tersebut kemudian dibagi menjadi dua permasalahan utama, yaitu pencarian menggunakan algoritma breadth-first search (BFS) dan algoritma depth-first search (DFS). Selain permasalahan utama tersebut, terdapat beberapa permasalahan kecil, yaitu membaca input file txt. Aplikasi harus dapat membaca file txt yang berisi maze dan dapat mengambil informasi mengenai posisi Krusty Krab (titik awal), posisi treasure, posisi grid yang dapat diakses, dan posisi grid yang tidak dapat diakses. Serta, mengkonversi maze ke dalam bentuk graf. Setelah mendapatkan informasi mengenai posisi-posisi penting di dalam maze, aplikasi harus dapat mengonversi maze tersebut ke dalam bentuk graf. Pada graf, setiap simpul merepresentasikan suatu grid dalam maze, dan setiap sisi merepresentasikan koneksi antara dua simpul yang dapat dilalui.

Selanjutnya, dilakukan pemetaan permasalahan ke dalam elemen-elemen algoritma BFS dan DFS, dan perancangan algoritma sesuai dengan elemen yang telah dipetakan. Setelah solusi dari pencarian menggunakan algoritma BFS dan DFS ditemukan, langkah selanjutnya adalah menampilkan solusi tersebut dalam bentuk desktop application dengan menggunakan bantuan WinForms. Program yang dibuat harus dapat menerima dan membaca input sebuah file txt yang berisi maze yang akan ditemukan solusi rute mendapatkan treasure-nya. Dengan mengikuti langkah-langkah pemecahan masalah tersebut, diharapkan program yang dibangun dapat memberikan solusi yang akurat dan efisien untuk memperoleh seluruh treasure atau harta karun yang ada pada maze.



### **3.2 Mapping Persoalan Menjadi Elemen-Elemen Algoritma BFS dan DFS**

Kita dapat memetakan elemen-elemen algoritma BFS dan DFS sebagai berikut:

1. State: State atau keadaan dalam kasus ini adalah posisi saat ini di dalam labirin atau maze.
2. Operator: Operator dalam kasus ini adalah gerakan untuk bergerak dari satu posisi ke posisi lainnya dalam maze, seperti naik, turun, kanan, dan kiri.
3. Goal state: Goal state atau keadaan tujuan dalam kasus ini adalah ketika seluruh harta karun dalam maze telah ditemukan.
4. Path cost: Path cost atau biaya jalur dalam kasus ini adalah jumlah langkah atau gerakan yang ditempuh untuk mencapai tujuan.

Dengan elemen-elemen tersebut, kita dapat memetakan proses algoritma BFS dan DFS sebagai berikut:

BFS (Breadth First Search):

1. Mula-mula, tentukan state awal (posisi saat ini) dan goal state (seluruh harta karun ditemukan).
2. Buat sebuah matrix yang akan mencatat state apa saja yang sudah dikunjungi.
3. Buat sebuah queue dan masukkan state awal ke dalam queue.
4. Setiap sebuah state akan dimasukkan ke dalam queue, catat state tersebut sebagai sudah dikunjungi.
5. Selama queue belum kosong, lakukan hal berikut:
  - Keluarkan state yang berada pada queue paling depan.
  - Jika state tersebut merupakan salah satu dari goal state:
    1. Catat jalur menuju goal state tersebut.
    2. Reset seluruh isi queue dan matrix kunjungan.
    3. Hapus goal state yang ditemukan dari kumpulan goal state.
    4. Ulangi algoritma dengan state awal merupakan goal state yang ditemukan.
  - Jika tidak, tambahkan state yang dapat dicapai dari state tersebut ke dalam queue.
6. Jika queue kosong dan goal state belum habis, maka tidak ada solusi.



7. Jika goal state sudah habis, maka terdapat solusi untuk mencapai seluruh harta karun.

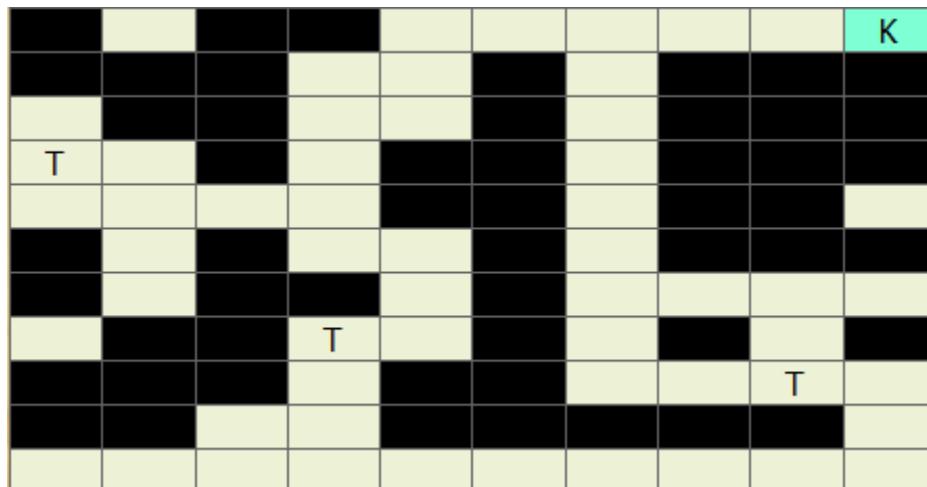
DFS (Depth First Search):

1. Mula-mula, tentukan state awal (posisi saat ini) dan goal state (seluruh harta karun ditemukan).
  2. Buat sebuah stack dan masukkan state awal ke dalam stack.
  3. Selama stack belum kosong, lakukan hal berikut:
    - Keluarkan state dari atas stack.
    - Jika pada state tersebut seluruh harta karun ditemukan, selesai dan kembalikan path.
    - Jika tidak, tambahkan state yang dapat dicapai dari state tersebut ke dalam stack.
  4. Jika stack kosong dan goal state belum ditemukan, maka tidak ada solusi.

Sebagai catatan, pemanggilan stack dapat disimulasikan dengan pemanggilan fungsi rekursif.

### 3.3 Ilustrasi Kasus Lain

Gambar 3.1 menunjukkan contoh maze pada persoalan tugas besar ini.



Gambar 3.1 Ilustrasi maze

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi Program

##### 4.1.1 BFS

```
Deklarasi Global
sizeX, sizeY: integer
petaAwal, peta: char[][][]
startCoordinate: Pair<int, int>
treasureCoordinate: Pair<int, int>[]
visited: boolean[][][]
canMove: boolean[][][][]
moveFrom: string[][][]
Move : string[] { prioritas arah dalam format string berisi (L, R, U, D) }

function BFS(input x: integer, input y: integer):string
{mengeluarkan langkah yang harus dilakukan apabila ingin mengunjungi seluruh
treasure dari titik awal (x,y)}
Deklarasi
    path, currPath : string
    searchQueue : Queue<Pair<int, int>>
    Start, curr, next : Pair<int, int>
    function reverse(input/output path: string)
    {membalikkan string yang diberikan}
    procedure softReset()
    {mereset seluruh nilai pada visited dan moveFrom}

Algoritma
    start <- Pair<int, int>(x, y)
    searchQueue.enqueue(start)
    visited[x][y] <- true
    while(searchQueue.size > 0 and treasureCoordinate.size > 0) do
        curr <- search.dequeue()
        i traversal [0..4]
        if(canMove[curr.first][curr.second][i] = false)continue
        next <- curr + moveDirection[move[i]]
        if(visited[next.first][next.second] = true)continue
        visited[next.first][next.second] <- true
        moveFrom[next.first][next.second] <- move[i]
        if(treasureCoordinate.contain(next)) then
            curr <- next
            currPath <- ""
            while(curr != start) do
                backMove <- moveFrom[curr.first][curr.second]
                currPath <- currPath + backMove
                curr <- curr - moveDirection[backMove]
                reverse(currPath)
                Path <- path + currPath
                treasureCoordinate.remove(next)

            softReset()
            searchQueue.clear()
            start <- next
```



```

    searchQueue.enqueue(start)
    Visited[start.first][start.second] <- true
  else
    searchQueue.enqueue(next)
-> path

```

#### 4.1.2 DFS

```

Deklarasi Global
peta: char[]
visited: boolean[]
needed: boolean[]
treasures: pair<int, int>[]
directPath: string { path dalam format string berisi (L, R, U, D) }
path: pair<int, int>[] { path dalam format array of koordinat }

function DFS(input x: integer, input y: integer, input prevDirection: char):
  boolean
{ Traversal peta dengan algoritma DFS }

Deklarasi
currentNodeNeeded: boolean

procedure Insert(input/output arr: Array of T, input newEl: T)
{ Meningkatkan ukuran efektif arr dan memasukkan newEl pada indeks
  terakhir arr }

function reverse(input direction: char): char
{ Mengembalikan char arah yang terbalik dari input.
  contoh:
    reverse('L') = 'R'
    reverse('D') = 'U' }

Algoritma
if (OutOfBounds(x, y))
  -> false
if (visited[x][y] or peta[x][y] = 'X' or AllTreasuresFound())
  -> false

  visited[x][y] <- true
  Insert(path, pair(x, y))
  Insert(directPath, prevDirection)

  currentNodeNeeded <- false

  for (char direction in {'L', 'R', 'U', 'D'})
    currentNodeNeeded <- currentNodeNeeded or DFS(x + dx(direction), y +
      dy(direction), direction)

  end for

  Insert(path, pair(x, y))
  Insert(directPath, reverse(prevDirection))
  needed[x][y] = currentNodeNeeded
-> currentNodeNeeded

```



## 4.2 Penjelasan Struktur Data

Struktur Data	Penggunaan
Pair	Struktur data buatan yang digunakan untuk merepresentasikan koordinat suatu titik pada peta
InputUtils	Struktur data buatan yang digunakan untuk mengambil, memvalidasi dan mengolah input menjadi data yang nantinya akan dipakai ketika melakukan traversal
List	Digunakan sebagai array dinamis ketika menyimpan kumpulan koordinat treasure, menyimpan kumpulan perubahan tahapan process ketika melakukan traversal
Queue	Digunakan pada BFS untuk membuat antrian koordinat yang akan diproses secara FIFO
StringBuilder	Digunakan sebagai alternatif string karena sifatnya yang mutable sehingga operasi concat ketika ingin mencatat path menjadi lebih efisien

## 4.3 Penjelasan Tata Cara Penggunaan Program

### 4.3.1 Prerequisite

Operating system: windows 10/ windows 11

Framework: .NET Core 7 (khusus build)

### 4.3.2 Build dan Run

- Clone repository pada komputer
- Buka direktori root pada repository
- Untuk build, jalankan

```
dotnet build src\WindowsFormsApp2
```

- Untuk build sekaligus run, jalankan

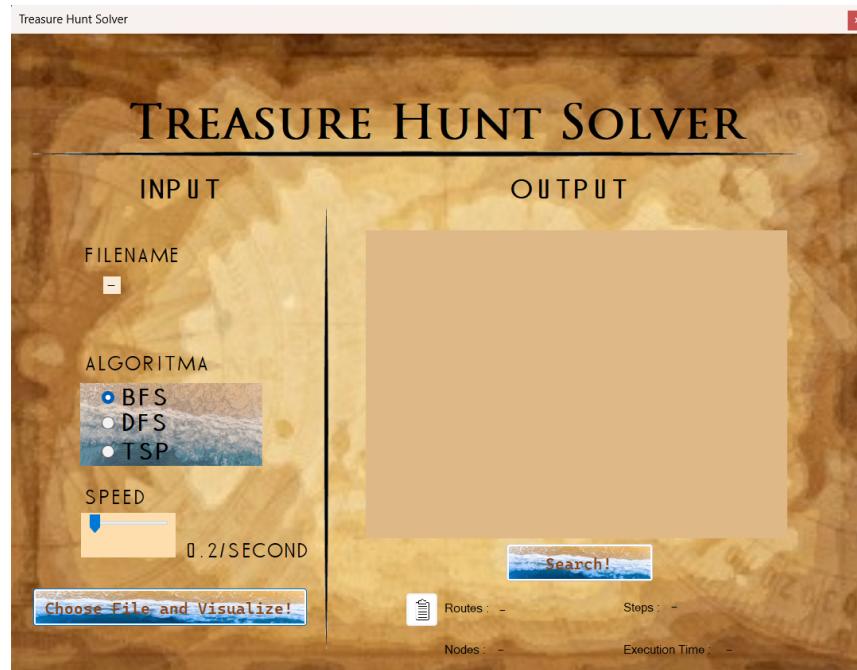
```
dotnet run --project src\WindowsFormsApp2
```

- Untuk run hasil compile, jalankan



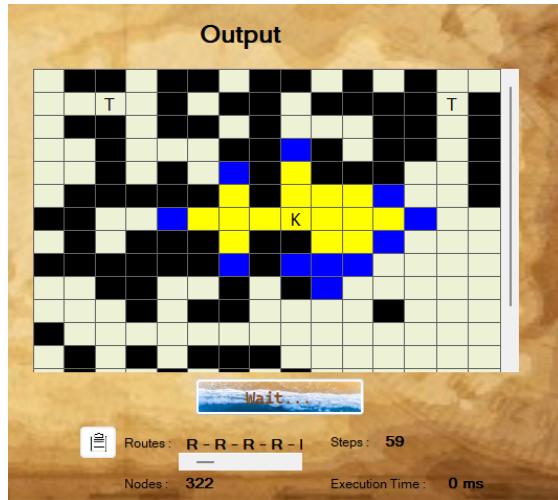
### 4.3.3 User Interface Program

Setelah melakukan perintah run, akan muncul program dengan tampilan seperti gambar 4.1. Sebelum melakukan visualisasi, pengguna harus menekan tombol “Choose File and Visualize” pada kiri bawah form, lalu memilih file input yang sesuai. Setelah itu, akan muncul tampilan peta pada bagian Output.



Gambar 4.1 Tampilan Program Awal

Setelah gambar peta muncul, pilih salah satu algoritma traversal di bagian kiri layar. Untuk menampilkan visualisasi, klik tombol “Search!”. Untuk mengubah kecepatan visualisasi, geser *slider* di bawah label “Speed”. Kecepatan meningkat jika *slider* semakin ke kanan; geser hingga ujung kanan untuk menghilangkan jeda pada visualisasi.



## Gambar 4.2 Tampilan Output

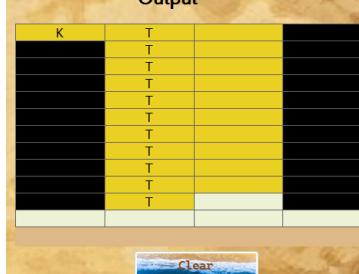
Pada visualisasi langkah traversal, program menggunakan warna biru untuk menandakan node(petak) sedang diproses, entah di dalam queue(bfs) atau pada pemanggilan rekursi(dfs). Setelah node selesai diproses, petak diberi warna kuning.

Pada visualisasi langkah akhir, program menggunakan warna kuning cerah untuk menandakan petak dikunjungi sekali. Petak yang dikunjungi dua kali diberi warna lebih gelap. Petak yang dikunjungi tiga kali atau lebih diberi warna lebih gelap.

Hasil eksekusi program (rute, step, node visited, execution time) terletak di bagian bawah Output seperti gambar 4.2. Terdapat tombol untuk menyalin hasil eksekusi program ke clipboard.

#### **4.4 Hasil Pengujian**

Input: sampel-5.txt

DFS	<p style="text-align: center;"><b>Output</b></p>  <p style="text-align: center;">Clear</p>	Execution time: 0 ms Node visited: 22 Steps: 21 Route: R - R - D - L - D - R - D - L - D - R - D - R - D - L - D - R - D - L - D
TSP	<p style="text-align: center;"><b>Output</b></p>  <p style="text-align: center;">Clear</p> <p>Routes: R - D - D - D - D - I Steps: 22 Nodes: 0 Execution Time: 10 ms</p>	Bruteforce Execution time: 77335 ms Node visited: 0 Steps: 22 Route: R - D - D - D - D - D - D - D - D - D - U - U - U - U - U - U - U - U - U - L  DP Execution time: 10 ms Node visited: 0 Steps: 22 Route: R - D - D - D - D - D - D - D - D - D - D - U - U - U - U - U - U - U - U - L

Input: maze20-5-1.txt

BFS	<p style="text-align: center;"><b>Output</b></p>  <p style="text-align: center;">Clear</p> <p>Routes: L - U - L - L - U Steps: 47 Nodes: 217 Execution Time: 12 ms</p>	Execution time: 12 ms Node visited: 217 Steps: 47 Route: L - U - L - L - U - L - D - D - R - D - D - L - D - R - U - R - R - R - R - R - R - D - R - D - U - L - U - U - U - R - R - R - U - L - U - L - L - U - U - L - U - L
-----	---	---

DFS	<p>Output</p> <p>Routes: L - U - L - L - U Steps: 71 Nodes: 102 Execution Time: 3 ms</p>	Execution time: 3 ms Node visited: 102 Steps: 71 Route: L - U - L - L - U - L - L - D - D - R - D - R - D - L - L - D - D - D - R - R - R - R - U - R - U - R - U - R - U - U - L - U - R - R - R - R - U - U - L - U - L - L - U - L - U - L - R - D - R - D - D - R - R - D - R - D - D - L - L - L - D - R - D - D - D - R - D
TSP	<p>Output</p> <p>Routes: R - D - R - D - I Steps: 70 Nodes: 0 Execution Time: 1 ms</p>	Execution time: 1 ms Node visited: 0 Steps: 70 Route: R - D - R - D - R - R - U - U - U - R - R - R - U - U - L - U - L - R - R - D - D - R - R - D - D - R - D - D - L - L - L - D - D - D - D - R - D - U - L - U - L - L - L - L - L - L - D - R - U - U - U - L - U - U - R - R - R - R - D - D - R

Input: maze-50-10.txt

BFS	<p>Output</p> <p>Routes: R - R - R - U - I Steps: 213 Nodes: 3659 Execution Time: 4 ms</p> <p>(View tidak dapat menampung seluruh peta)</p>	Execution time: 4 ms Node visited: 3659 Steps: 213 Route: R - R - R - U - R - R - D - R - R - R - R - R - R - R - R - R - R - R - U - R - R - R - R - U - U - R - R - R - R - U - U - U - U - U - U - D - R - D - D - D - L - D - D - D - D - D - D - L - D - D - D - D - D - R - ... (Route dipotong karena terlalu panjang)
-----	---	---

DFS		Execution time: 1 ms Node visited: 1500 Steps: 800 Route: L - L - L - L - L - U - L - L - L - U - L - L - L - L - L - U - R - U - L - L - U - L - L - L - U - L - U - L - U - R - R - R - R - D - R - D - D - R - R - R - R - ...
TSP		Execution time: 12 ms Node visited: 0 Steps: 244 Route: L - L - L - L - L - U - L - L - L - U - L - L - L - L - L - U - U - U - U - U - U - U - U - U - U - R - U - U - U - L - U - R - D - D - D - D - L - D - D - D - D - D - L - L - D - D - L - D - D - D - ...

#### 4.5 Analisis Desain Algoritma

Dalam segi banyaknya node yang akan dikunjungi ketika melakukan traversal bfs akan mengunjungi lebih banyak node dibanding dengan dfs, banyaknya kunjungan node juga membuat waktu eksekusi bfs lebih lama dibandingkan dfs.

Dalam segi langkah rata-rata yang dibutuhkan untuk mengunjungi seluruh harta karun, bfs akan memiliki langkah yang lebih kecil daripada dfs karena bfs merupakan dasar dari algoritma shortest path dijkstra dan memang merupakan algoritma untuk mencari shortest path pada graph berbobot 1, meskipun hasilnya belum optimal karena pencarian masih bersifat greedy bukan mencoba seluruh urutan pengambilan seperti bruteforce.

Pencarian dfs akan menghasilkan langkah yang sangat buruk apabila kita perlu berjalan hingga ke ujung meskipun di jalan tersebut tidak ada harta karun, dan ketika kita sudah menemukan harta karun, path yang didapat tetap harus mengikuti jalan panjang tersebut sehingga langkah yang dihasilkan akan menjadi sangat panjang. Selain itu karena pemilihan prioritas arah traversal tetap, ketika peta dirotasi akan menghasilkan hasil yang berbeda juga.

Dalam perhitungan di bawah, misal  $r = \text{jumlah baris}$ ,  $c = \text{jumlah kolom}$ , dan  $t = \text{jumlah treasure}$

Kompleksitas algoritma DFS:

Jumlah eksekusi = 1

Jumlah node =  $rc$

Jumlah edge  $\approx 2rc$

Sehingga didapat kompleksitas akhir

$$T(n) = O(rc + 2rc)$$

$$T(n) = O(rc)$$

Kompleksitas algoritma BFS:

Jumlah eksekusi = t

Jumlah node =  $rc$

Jumlah edge  $\approx 2rc$

Sehingga didapat kompleksitas akhir

$$T(n) = O(t(rc + 2rc))$$

$$T(n) = O(trc)$$

Perbandingan jumlah node yang dikunjungi dan jumlah langkah

Nama file	Algoritma	Node yang dikunjungi	Jumlah langkah
sample-5.txt	BFS	43	11
	DFS	22	21
maze10-3-1.txt	BFS	89	43
	DFS	40	44
maze10-3-2.txt (rotasi dari maze10-3-1.txt)	BFS	92	43
	DFS	39	45
maze20-5-1.txt	BFS	217	47
	DFS	102	71
maze20-5-2.txt (rotasi dari maze20-5-1.txt)	BFS	222	49
	DFS	107	100
maze50-10.txt	BFS	3659	213
	DFS	1500	800
maze50-15.txt	BFS	6251	233
	DFS	2291	3131

## **BAB V**

### **KESIMPULAN, SARAN, DAN REFLEKSI**

Dari Tugas Besar II IF2211 Strategi Algoritma Semester II 2022/2023 berjudul *Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt*, kami mendapatkan bahwa untuk mencari jalan keluar dari sebuah labirin dapat dilakukan dengan menggunakan pendekatan strategi *BFS* dan *DFS*. Namun, masing-masing algoritma memiliki kelebihan dan kekurangan yang perlu dipertimbangkan tergantung pada sifat dari masalah yang akan diselesaikan.

Saran-saran yang dapat kami berikan untuk Tugas Besar II IF2211 Strategi Algoritma Semester II 2022/2023 adalah, seperti yang telah dibahas sebelumnya, baik *BFS* maupun *DFS* memiliki kelebihan dan kekurangan masing-masing. Sebelum memilih algoritma yang akan digunakan, pertimbangkan sifat dari masalah yang akan diselesaikan seperti ukuran labirin, kompleksitas struktur labirin, dan jumlah solusi yang mungkin. Dalam beberapa kasus, salah satu algoritma mungkin lebih efektif daripada yang lain. Meskipun *DFS* dan *BFS* cukup efektif dalam menyelesaikan persoalan *Maze Treasure Hunt*, masih ada ruang untuk optimalisasi. Selain itu, sebagai bagian dari penelitian lebih lanjut, disarankan untuk melakukan perbandingan terhadap variasi algoritma *DFS* dan *BFS* yang dapat digunakan untuk menyelesaikan persoalan *Maze Treasure Hunt*. Beberapa contohnya adalah *DFS* dengan backtracking, *DFS* dengan branch-and-bound, atau *BFS* dengan heuristik. Hal ini dapat memberikan gambaran yang lebih jelas mengenai kelebihan dan kekurangan dari masing-masing algoritma.

Setelah menyelesaikan Tugas Besar II IF2211 Strategi Algoritma Semester II 2022/2023, kami dapat merefleksikan bahwa komunikasi antaranggota kelompok berjalan cukup baik sehingga tidak terjadi miskomunikasi dan kesalahpahaman dalam pengerjaan tugas kecil ini. Sebelum dimulainya pengerjaan tugas besar ini juga kami melakukan diskusi untuk membahas pembagian kerja untuk setiap anggota kelompok. Oleh karena itu, kami mendapatkan hasil yang sesuai dengan tujuan dan sasaran yang telah ditetapkan pada awal pengerjaan.

## **Daftar Pustaka**

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>

<https://www.kodesiana.com/post/mengatasi-lag-datagridview-csharp/>

<https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/>

<https://www.youtube.com/watch?v=cY4HiiFHO1o>



## **Lampiran**

Repositori program: [https://github.com/Michaelu670/Tubes2\\_okebebasya](https://github.com/Michaelu670/Tubes2_okebebasya)  
Link video : <https://youtu.be/c39neI8JOj4>

