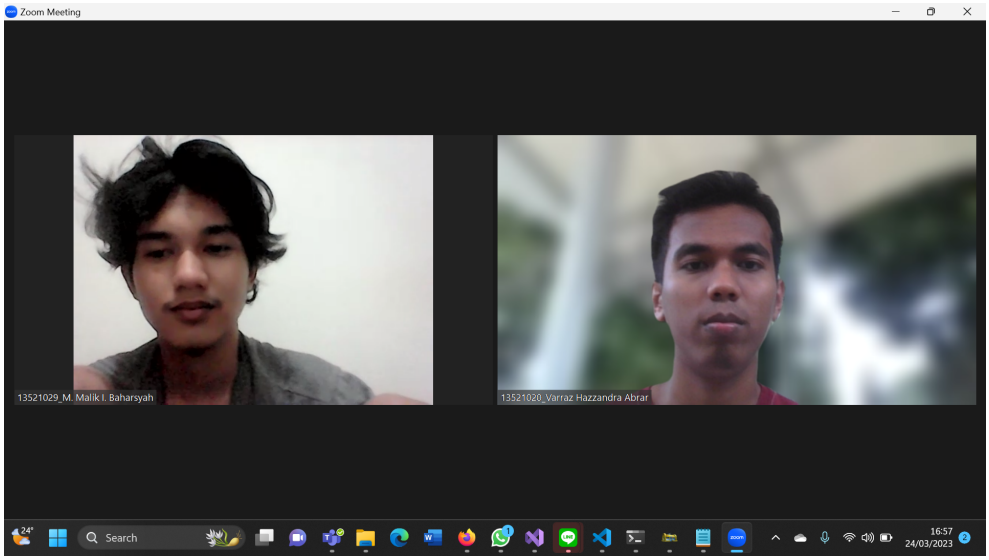
LAPORAN TUGAS BESAR 1 IF2211 STRATEGI ALGORITMA PENGAPLIKASIAN ALGORITMA BFS DAN DFS DALAM MENYELESAIKAN PERSOALAN MAZE TREASURE HUNT



Oleh:

Kelompok "Kepiting Matre"

Varraz Hazandra Abrar	13521020
M. Malik I. Baharsyah	13521029

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

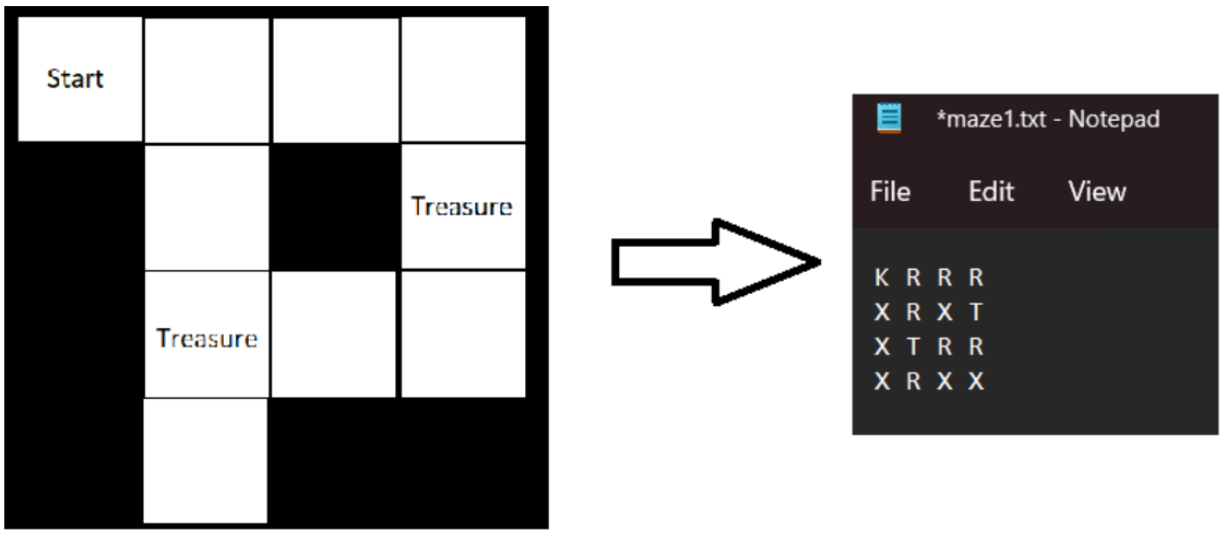
BAB I
DESKRIPSI MASALAH

1.1 Deskripsi Masalah

Sebuah aplikasi dengan GUI sederhana yang dapat mengimplementasikan BFS dan DFS dibutuhkan untuk mendapatkan rute memperoleh seluruh treasure atau harta karun yang ada. Program dapat menerima dan membaca input sebuah file txt yang berisi maze yang akan ditemukan solusi rute mendapatkan treasure-nya. Untuk mempermudah, batasan dari input maze cukup berbentuk segi-empat dengan spesifikasi simbol sebagai berikut :

- K : Krusty Krab (Titik awal)
- T : Treasure
- R : Grid yang mungkin diakses / sebuah lintasan
- X : Grid halangan yang tidak dapat diakses

Contoh file input :



Gambar 1 Ilustrasi Input File Maze

BAB II
LANDASAN TEORI

2.1 Graf Traversal

Traversal graf adalah algoritma yang mana mengunjungi simpul dengan cara yang sistematik, yaitu dengan pencarian melebar (breadth first search/BFS) dan pencarian mendalam (depth first search/DFS). Diasumsikan bahwa graf terhubung. Graf bisa direpresentasikan sebagai persoalan dan traversal graf sebagai pencarian solusi.

2.2 Algoritma BFS

Algoritma breadth-first search (BFS) adalah algoritma yang digunakan untuk mencari struktur data pohon atau grafik untuk sebuah node yang memenuhi serangkaian kriteria. Algoritma ini dimulai dari simpul akar dan mengunjungi semua node pada tingkat kedalaman saat ini sebelum beralih ke node pada tingkat kedalaman berikutnya. Hal yang perlu diperhatikan di sini adalah tidak seperti pohon, graf dapat mengandung siklus sehingga kita dapat kembali ke node yang sama. Untuk menghindari pemrosesan sebuah node lebih dari sekali, kita membagi node-node ke dalam dua kategori: visited dan not visited. Sebuah larik boolean visited digunakan untuk menandai node yang dikunjungi. Untuk mempermudah, diasumsikan bahwa semua node dapat dijangkau dari node awal. BFS menggunakan struktur data queue untuk penjelajahan.

2.3 Algoritma DFS

Algoritma Depth-first search (DFS) adalah sebuah algoritma yang digunakan menelusuri atau mencari struktur data pohon atau grafik. Algoritma ini dimulai dari simpul akar (memilih beberapa simpul sembarang sebagai simpul akar dalam kasus graf) dan menjelajahi sedalam mungkin di sepanjang setiap cabang sebelum melakukan penelusuran balik. Ide dasarnya adalah memulai dari akar atau simpul sembarang dan menandai simpul tersebut dan pindah ke simpul yang tidak ditandai dan melanjutkan perulangan ini hingga tidak ada simpul yang tidak ditandai yang berdekatan. Lalu lakukan *backtracking* dan periksa node-node lain yang belum ditandai dan telusuri node-node tersebut.

2.4 C# Desktop Application Development

C# Desktop Application Development adalah proses membuat aplikasi desktop menggunakan bahasa pemrograman C# (C Sharp) dan framework .NET. Dalam pengembangan aplikasi desktop C#, developer dapat menggunakan berbagai teknologi seperti Windows Forms, Windows Presentation Foundation (WPF), atau Universal Windows Platform (UWP).

Dalam pengembangan C# Desktop Application, developer dapat membuat aplikasi desktop yang dapat dijalankan pada sistem operasi Windows dengan menggunakan teknologi-teknologi tersebut. Aplikasi desktop C# dapat memiliki berbagai fitur seperti antarmuka pengguna, database, akses ke jaringan, pencetakan dokumen, dan lain sebagainya.

Pengembangan aplikasi desktop menggunakan C# memungkinkan developer untuk membangun aplikasi dengan cepat dan efisien, karena C# adalah bahasa pemrograman modern dan memiliki fitur-fitur yang lengkap. Selain itu, C# juga memiliki integrasi yang baik dengan lingkungan pengembangan Visual Studio, sehingga developer dapat membuat aplikasi desktop dengan mudah dan cepat.

BAB III
APLIKASI BFS DAN DFS

3.1 Langkah-langkah pemecahan masalah

Berikut adalah langkah-langkah pemecahan masalah untuk algoritma pencarian rute terpendek untuk mendapatkan semua harta karun yang terdapat dalam grid/maze:

1. Membaca inputan file txt yang merepresentasikan maze dan menentukan koordinat start dan treasure jika ada.
2. Membuat queue yang merepresentasikan apakah sebuah node sudah dikunjungi.
3. Memasukkan node awal, yaitu start ke dalam queue
4. Selama queue masih berisi node yang belum dikunjungi, langkah-langkahnya adalah sebagai berikut.

- BFS

    a. Keluarkan node dari depan queue.
    b. Jika node tersebut adalah treasure, pencarian selesai dan kembalikan jalur yang ditemukan.
    c. Jika node tersebut bukan treasure, cari node-node tetangganya yang belum dikunjungi.
    d. Untuk setiap node-node tetangga yang belum dikunjungi, kunjungi dan tandai node tersebut sebagai sudah dikunjungi dan masukkan ke dalam queue dengan menyertakan jalur yang telah ditempuh untuk mencapai node tersebut.
    e. Simpan jalur terpendek yang telah ditemukan.

- DFS

    a. Keluarkan node dari depan queue.
    b. Jika node tersebut adalah treasure, pencarian selesai dan kembalikan jalur yang ditemukan.
    c. Jika node tersebut bukan treasure, telusuri cabang dari node sampai tidak ada lagi node yang bisa dikunjungi. Jika sudah tidak ada lagi, mundur ke node paling awal tadi dan jelajahi node tetangga yang belum dikunjungi. Lakukan langkah b dan c untuk tiap node.
    d. Untuk setiap node-node tetangga yang belum dikunjungi, kunjungi dan tandai node tersebut sebagai sudah dikunjungi dan masukkan ke dalam queue dengan menyertakan jalur yang telah ditempuh untuk mencapai node tersebut.
    f. Simpan jalur terpendek yang telah ditemukan.

5. Jika queue kosong dan jalur terpendek belum ditemukan, tidak ada jalur yang dapat ditempuh untuk mencapai treasure.

3.2 Mapping Persoalan

Berikut mapping persoalan ke implementasi algoritma BFS dalam persoalan Maze Treasure Hunt.

1. Pohon ruang status : titik-titik ditandai dengan simpul yang diberi warna sebagai penanda titik mana saja yang sudah dikunjungi dan tidak dikunjungi. Lakukan ke semua simpul di level yang sama untuk kemudian ke simpul di level selanjutnya.
2. Simpul : terbagi menjadi dua :
    a. Akar : titik start
    b. Daun : semua titik yang bisa dikunjungi yang menjadi rute pada maze.
3. Cabang : operator enque ke dalam queue yang berisi simpul yang telah ditelusuri
4. Ruang status : seluruh simpul di dalam pohon ruang status algoritma BFS
5. Ruang solusi : himpunan dari semua jalur yang menuju ke suatu titik. Solusi berupa rute yang menghubungkan start ke titik keluar maze dengan melalui semua titik treasure.

Berikut mapping persoalan ke implementasi algoritma BFS dalam persoalan Maze Treasure Hunt.

1.Pohon ruang status : titik-titik ditandai dengan simpul yang diberi warna sebagai penanda titik mana saja yang sudah dikunjungi dan tidak dikunjungi. Lakukan sampai tidak ada lagi simpul yang bisa dikunjungi lalu beralih ke simpul tetangga yang belum dikunjungi.
2. Simpul : terbagi menjadi dua :
    a. Akar : titik start
    b. Daun : semua titik yang bisa dikunjungi yang menjadi rute pada maze.
3. Cabang : operator add ke dalam list yang berisi simpul yang telah ditelusuri yang tersusun sesuai aturan DFS
4. Ruang status : seluruh simpul di dalam pohon ruang status algoritma DFS.
5. Ruang solusi : himpunan dari semua jalur yang menuju ke suatu titik. Solusi berupa rute yang menghubungkan start ke titik keluar maze dengan melalui semua titik treasure.

3.3 Contoh Kasus Lain

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

Berikut adalah pseudocode untuk algoritma pencarian rute terpendek untuk mendapatkan seluruh harta karun dalam maze yang telah dibuat:

1. **BFS**

```
Procedure FindShortestRoute() -> string
KAMUS LOKAL
elapsedTime: Stopwatch
queue: Queue
visited: boolean[]
node: Node
startX: integer
startY: integer
neighbors: Node[]
countNodes: integer
ALGORITMA
elapsedTime.Start()
queue(Enqueue( Node(startX, startY, “”, 0))
visited[startX, startY, 0] <- true
while queue.Count > 0 do
   node <- queue.Dequeue()
   if IsFinalState(node) then
      elapsedTime.Stop()
      -> node.path
   neighbors <- GetNeighbors(node)
   i traversal [0..neighbors.Count]
      if not visited[neighbors[i].x, neighbors[i].y,
neighbor.collectedTreasure] then
         visited[neighbors[i].x, neighbors[i].y,
neighbor.collectedTreasure] <- true
         queue.Enqueue(neighbor)
         countNodes <- countNodes + 1
elapsedTime.Stop()
-> “No route found”
```

2. **DFS**

```
Procedure FindShortestRoute() -> string
KAMUS LOKAL
elapsedTime: Stopwatch
shortestRoute: string
start: (integer, integer)
rows: integer
cols: integer
tempBool: bool[,]
ALGORITMA
tempBool <- bool[rows, cols]
```

```
elapsedTime.Start()
DFS(start[0], start[1], "", 0)
elapsedTime.Stop()
-> shortestRoute

Procedure DFS(input row,col: integer, input route: string, input
visited: boolean[,], input steps: integer)
KAMUS LOKAL
grid: char[,]
row: integer
col: integer
rows: integer
cols: integer
visited: boolean[,]
nodesChecked: integer
treasureCount: integer
steps: integer
minSteps: integer

ALGORITMA
if row < 0 or row >= rows or col < 0 or col >= cols or
visited[row,col] = true or grid[row, col] = 'X' then
   -> ""
visited[row,col] <- true
nodesChecked <- nodesChecked + 1
If grid[row, col] = 'T' then
   treasureCount <- treasureCount - 1
if treasureCount = 0 then
   if steps < minSteps then
      minSteps <- steps
      shortestRoute <- route
   visited[row,col] <- false
   treasureCount <- treasureCount + 1
   -> ""
DFS(row + 1, col, route + "D", visited, steps + 1)
DFS(row - 1, col, route + "U", visited, steps + 1)
DFS(row, col + 1, route + "R", visited, steps + 1)
DFS(row, col - 1, route + "L", visited, steps + 1)

visited[row,col] <- false
if grid[row,col] = 'T' then
   treasureCount <- treasureCount + 1
```

4.2 Struktur Data dan Spesifikasi Program

Struktur dan tipe data yang digunakan dalam implementasi Maze adalah sebagai berikut.

1. List<Tuple<int, int>>

Struktur data list menyimpan tuple yang berisi koordinat harta karun.

2. Queue<Node>

Struktur data queue menyimpan simpul-simpul yang sudah dikunjungi.

3. FileInfo

Tipe data FileInfo digunakan untuk melihat atribut sebuah file, misalnya nama file.

4. Stopwatch

Tipe data ini melakukan perhitungan waktu eksekusi program.

Program memiliki kelas :

1. GetFile

Method	**Keterangan**
public static char[,] GetGrid(string path)	Method ini memiliki fungsi yang mengolah grid dari sebuah input path file .txt dan akan mereturn sebuah matriks grid

2. GridOperations

Method	**Keterangan**
public static List<(int, int)> getTreasuresCoord(char[,] inputGrid)	Method untuk mereturn sebuah list of koordinat dari treasures yang ada di grid
public static List<(int, int)> getJellyFishCoord(char[,] inputGrid)	Method untuk mereturn sebuah list of koordinat dari grid yang tidak dapat diakses atau *obstacle*
public static (int, int) getStartCoord(char[,] inputGrid)	Method untuk mereturn sebuah koordinat *starting point* dari grid
public static bool IsValidGrid(char[,] inputGrid)	Method untuk validasi apakah sebuah grid terdapat karakter selain K, R, X, T
public static bool IsValidChar(char c)	Method untuk validasi apakah sebuah karakter tidak terdapat di (K, R, X, T)

3. TreasureHunterBFS

Atribut	**Keterangan**
private char[,] grid	Atribut untuk menyimpan grid
private int startX, startY	Atribut untuk menyimpan starting point
private List<Tuple<int, int>> treasures	Atribut untuk menyimpan list of koordinat treasures
private Queue<Node> queue	Atribut untuk menyimpan queue yang nantinya digunakan di algoritma BFS
private bool[,,] visited	Atribut untuk menyimpan boolean sebuah point sudah dilewati atau belum

public Stopwatch elapsedTime	Atribut untuk menyimpan waktu eksekusi
public int countNodes	Atribut untuk menyimpan jumlah node yang telah dicek

Method	**Keterangan**
public TreasureHunterBFS(char[,] grid)	Method konstruktor TreasureHunterBFS
public string FindShortestRoute()	Method yang berfungsi untuk mereturn sebuah string yang berisi rute terpendek dalam pencarian seluruh harta karun yang ada dalam maze
private bool IsFinalState(Node node)	Method yang berfungsi mereturn node adalah state final
private List<Node> GetNeighbors(Node node)	Method yang berfungsi mereturn list node yang bertetanggaan dengan node tertentu
private int UpdateCollectedTreasures(Node node, int x, int y)	Method yang berfungsi mereturn jumlah treasure yang telah didapat

4. TreasureHunterDFS

Atribut	**Keterangan**
private char[,] grid	Atribut untuk menyimpan grid
private int rows	Atribut untuk menyimpan jumlah baris
private int cols	Atribut untuk menyimpan jumlah kolom
private int treasureCount	Atribut untuk menyimpan jumlah treasures
private (int row, int col)	Atribut untuk menyimpan koordinat yang sedang berlangsung
private List<(int row, int col)> treasures	Atribut untuk menyimpan list of koordinat treasure yang ada
private string shortestRoute	Atribut untuk menyimpan string berupa rute terpendek yang didapat
private int minSteps	Atribut untuk menyimpan integer step paling minim yang dilakukan
private int nodesChecked	Atribut untuk menyimpan jumlah node yang telah dicek
private Stopwatch elapsedTime	Atribut untuk menyimpan waktu eksekusi

Method	**Keterangan**
public TreasureHunterDFS(char[,] grid)	Method konstruktor
public string FindShortestRoute()	Method yang berfungsi mereturn string berupa rute terpendek
public int GetNodes()	Method yang berfungsi mereturn jumlah node yang telah dicek
public long GetElapsedTime()	Method yang berfungsi mereturn waktu eksekusi
private void DFS(int row, int col, string route, bool[,] visited, int steps)	Method yang merupakan algoritma utama dari DFS

4.3 Tata Cara Penggunaan Program

Berikut cara penggunaan program (sudah dalam bentuk aplikasi siap pakai).

1.Unduh atau clone repositori ini di terminal

2. Kunjungi bin pada repositori untuk menemukan file TreasureHunt.exe

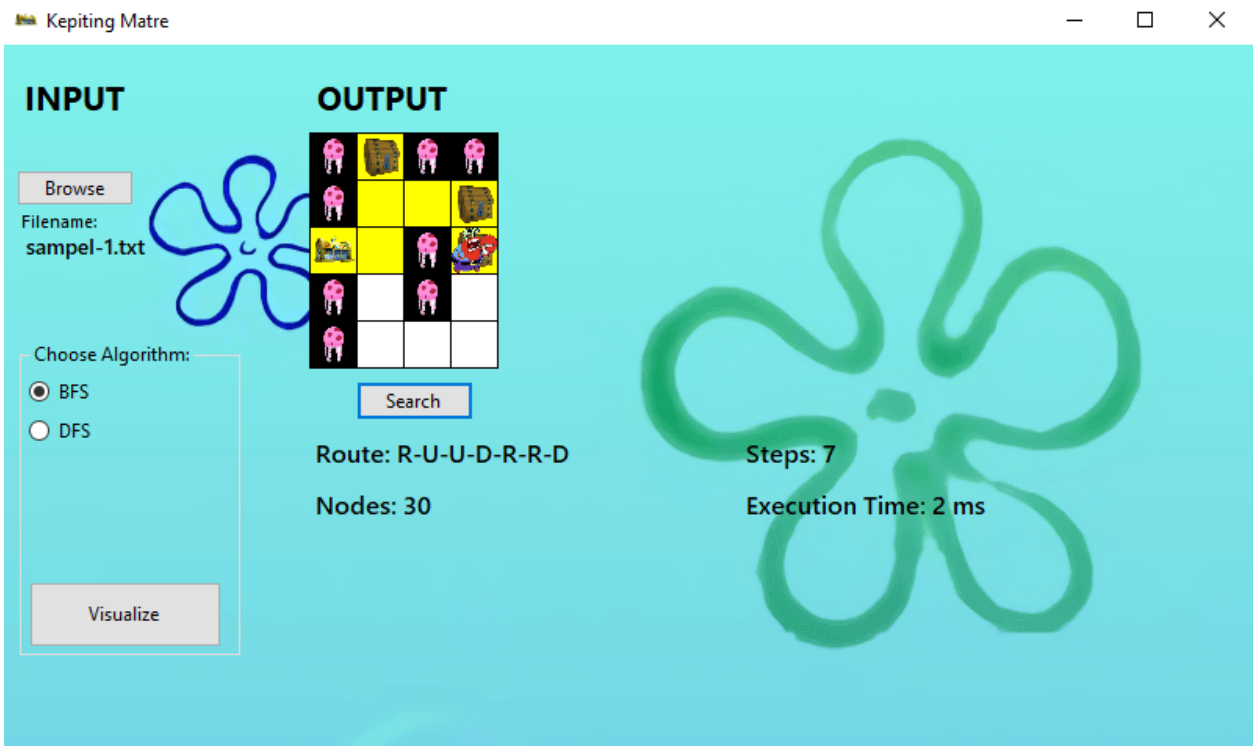
3. Klik kiri 2 kali pada file tersebut dan aplikasi siap digunakan

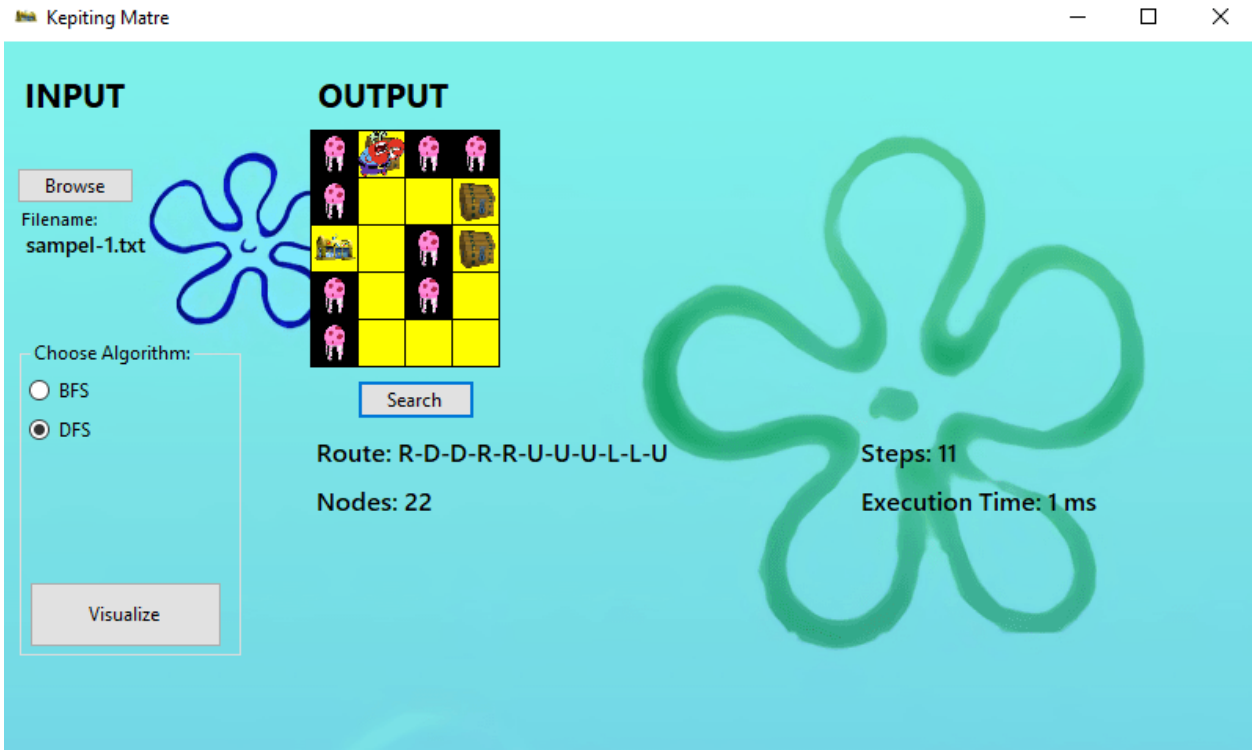
4.4 Hasil Pengujian

Setelah dilakukan pengetesan program berdasarkan 5 sampel input file berformat txt, didapatkan input dan output dari masing-masing algoritma BFS dan DFS sebagai berikut:

1. **Sampel 1**
   Input Grid:
   X T X X
   X R R T
   K R X T
   X R X R
   X R R R

   Tampilan Output:

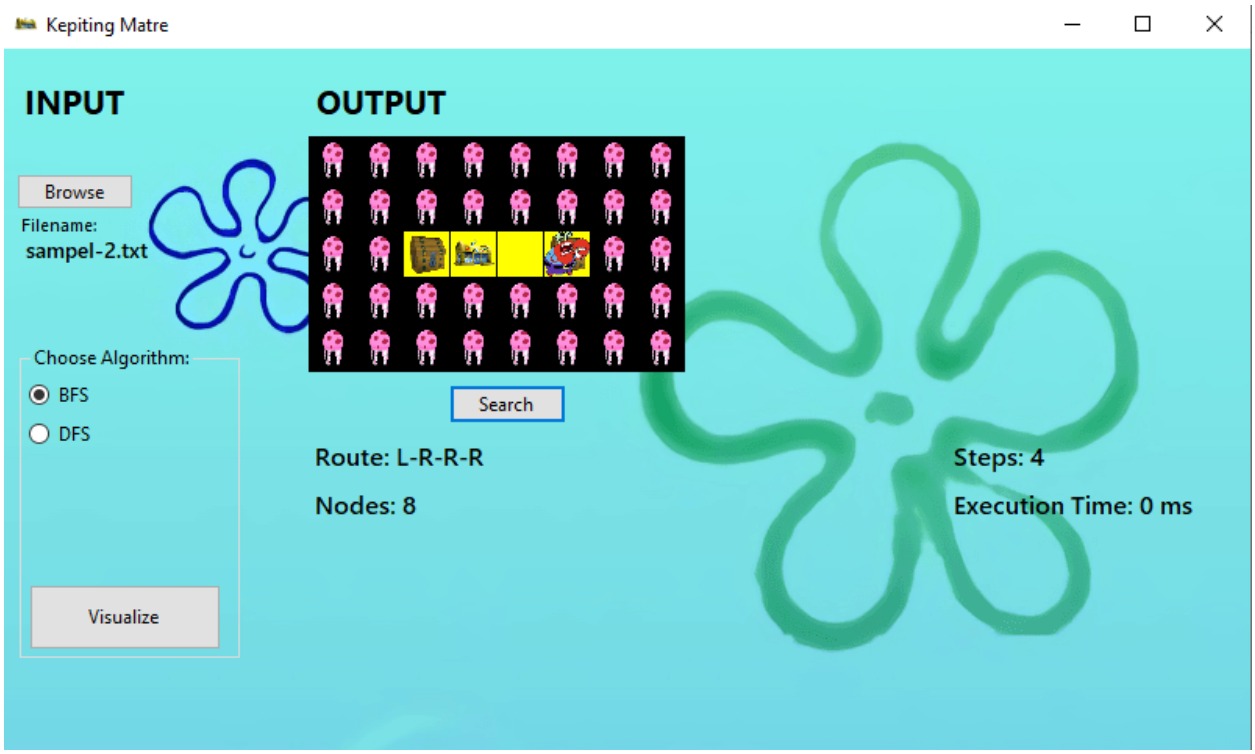


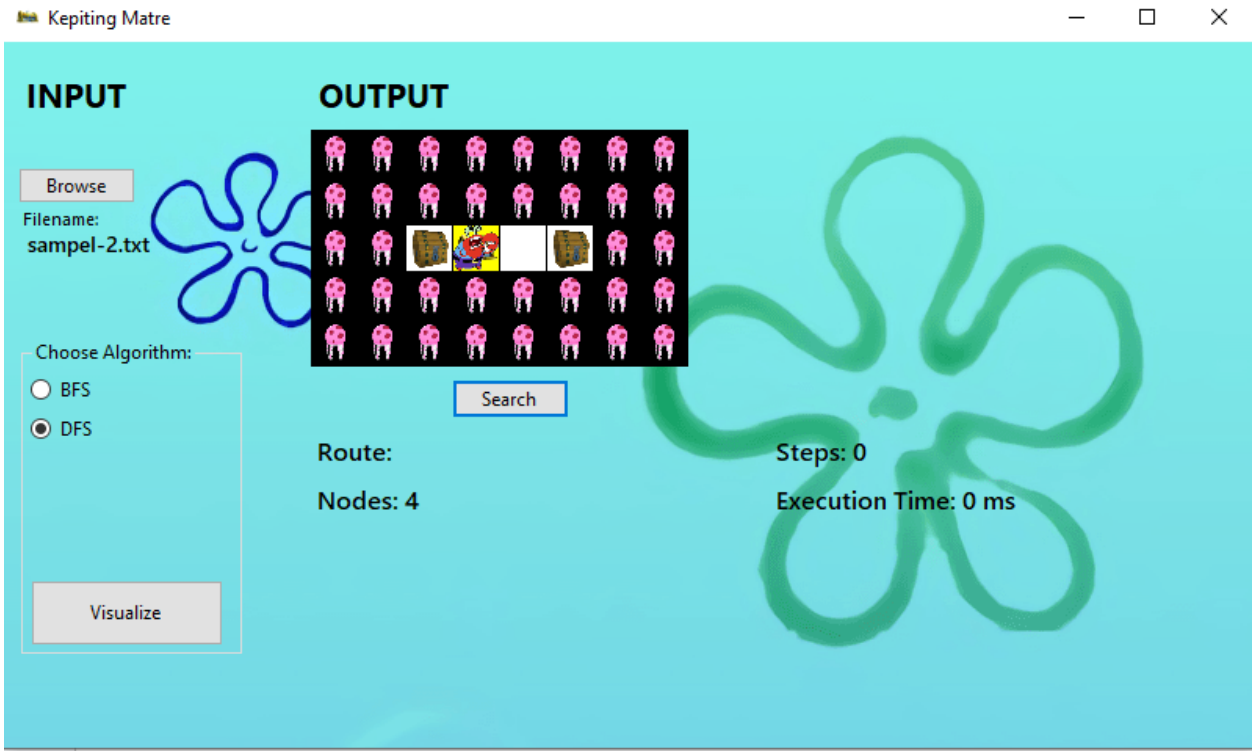


2. **Sampel 2**
   Input Grid:
   X X X X X X X X
   X X X X X X X X

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

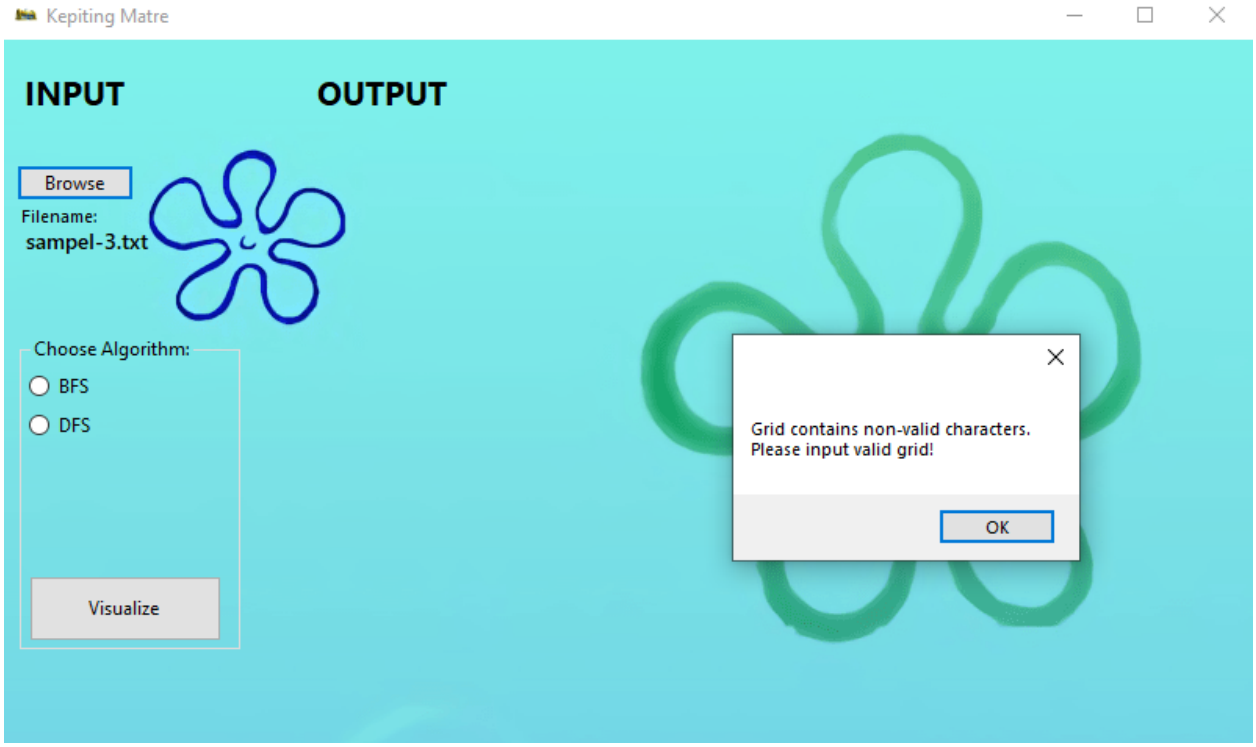
Tampilan Output:





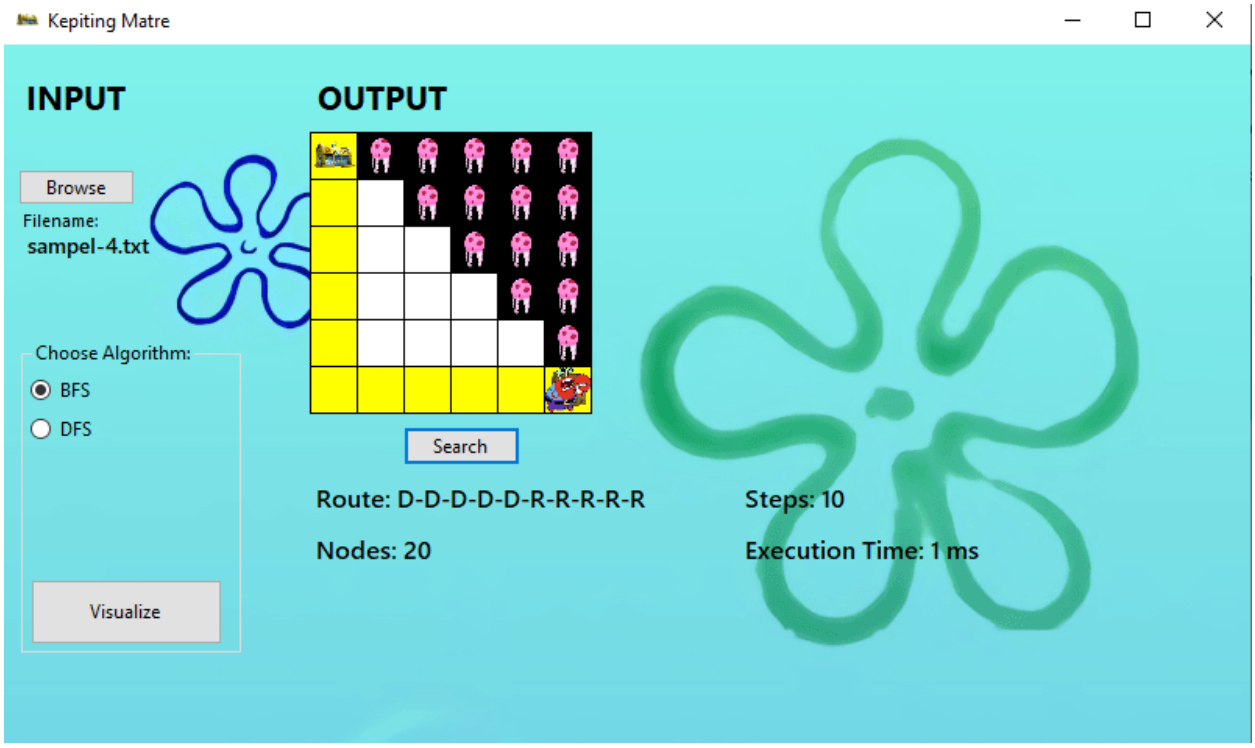
3. **Sampel 3**
   Input Grid:
   J A N G A N
   L U P A C E
   K Y A N G B
   E G I N I Y

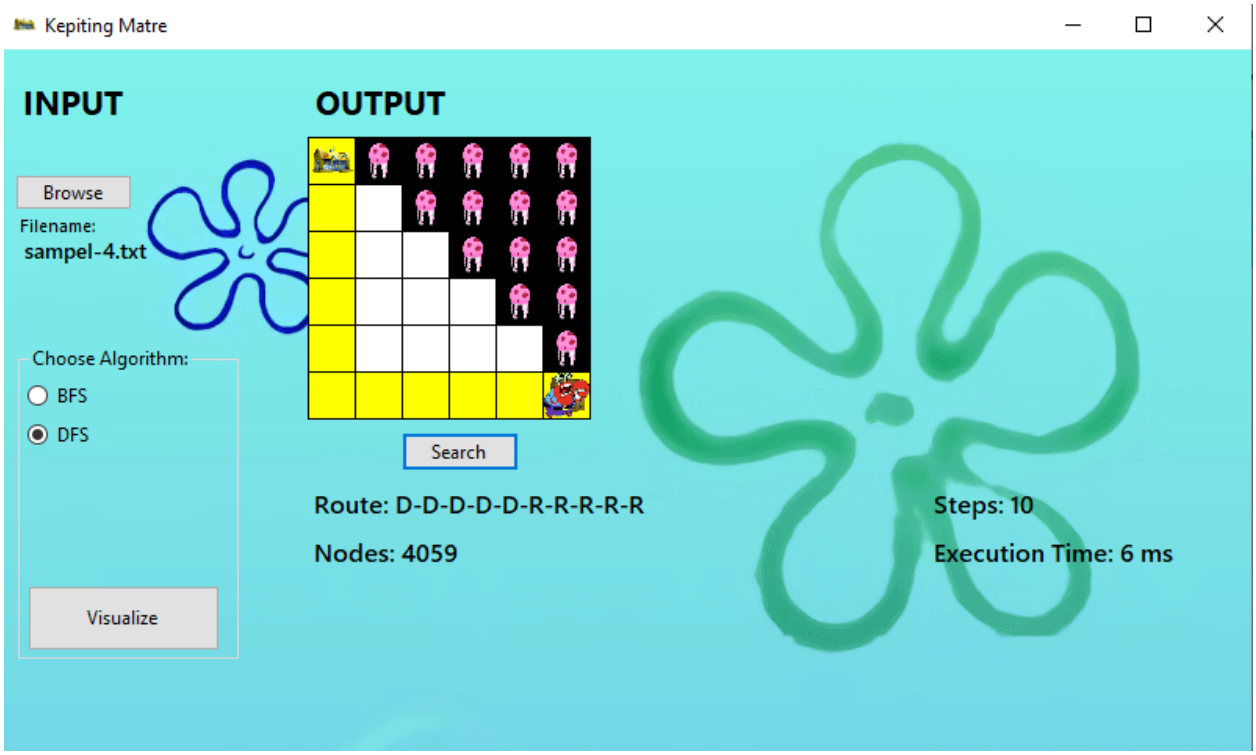
   Tampilan Output:



4. **Sampel 4**
   Input Grid:
   K X X X X X
   R R X X X X
   R R R X X X
   R R R R X X
   R R R R R X
   R R R R R T

Tampilan Output:

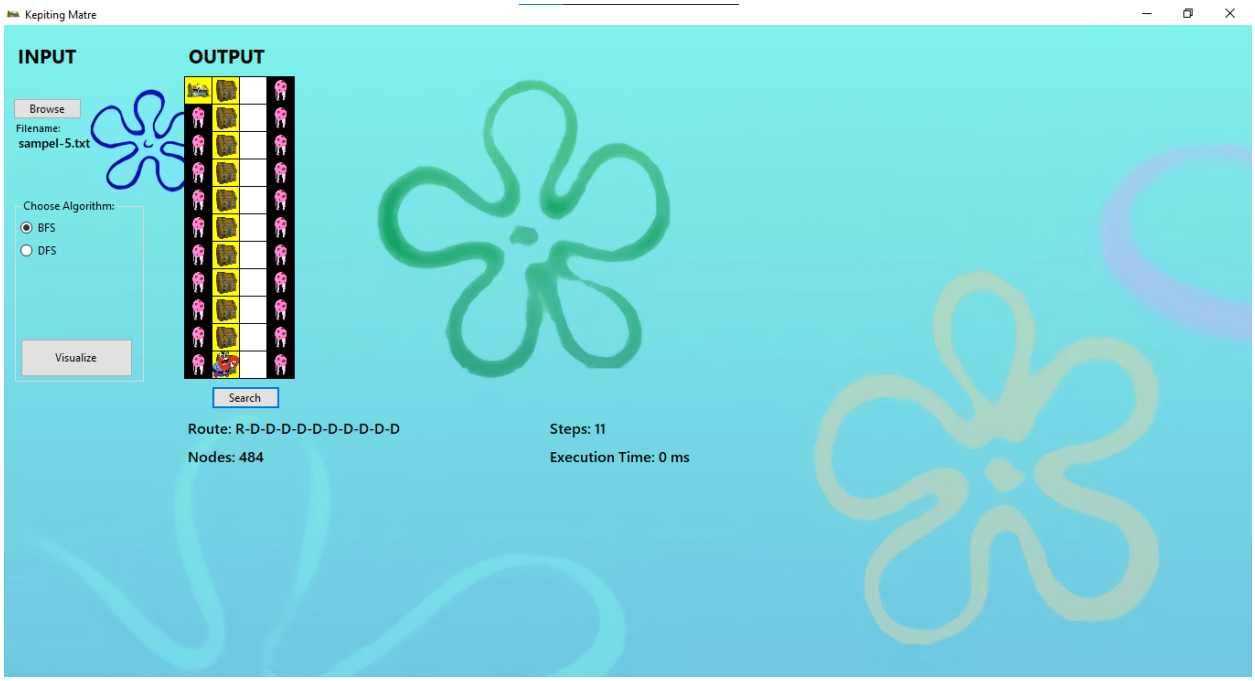


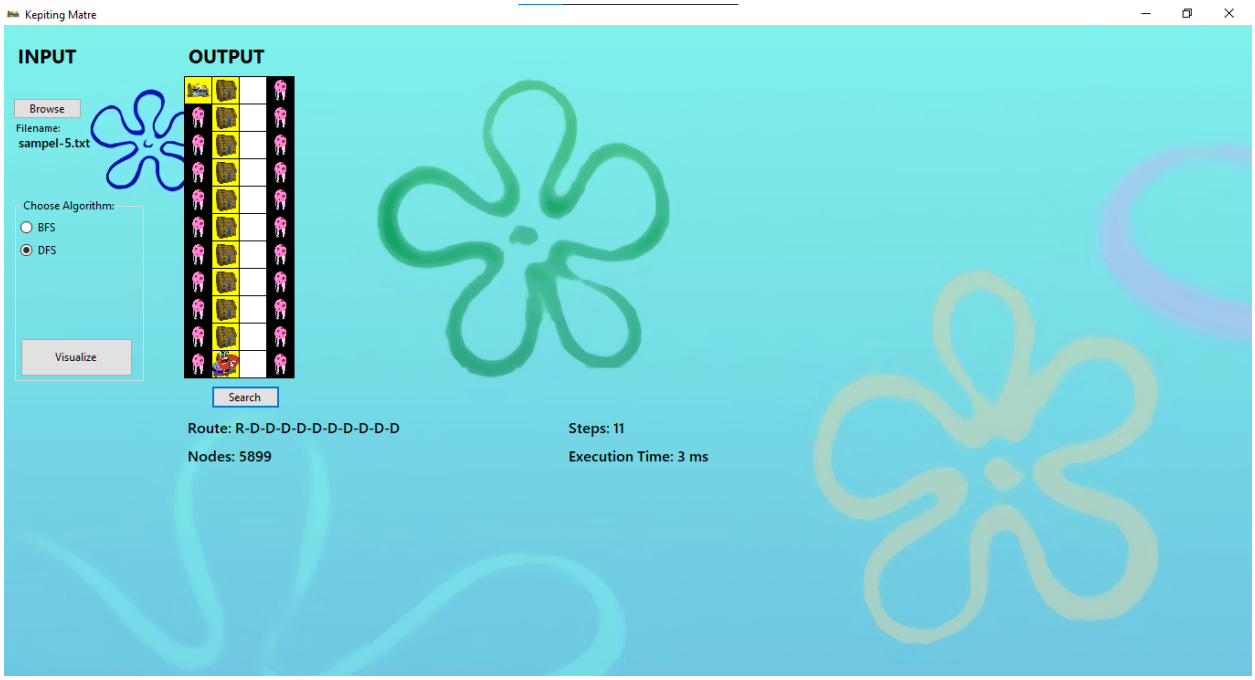


5. **Sampel 5**
   Input Grid:
   K T R X

X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X

Tampilan Output:





4.5 Analisis

Berdasarkan Algoritma BFS dan DFS yang telah dibuat untuk mencari rute terpendek dalam pencarian seluruh harta karun yang ada di maze, algoritma DFS memiliki jumlah node yang dikunjungi cenderung memiliki angka yang jauh lebih banyak daripada algoritma BFS. Hal tersebut dikarenakan algoritma DFS mengunjungi setiap ujung node terlebih dahulu dan dilakukan *backtracking*. Pengunjungan node yang lebih banyak berbanding lurus terhadap *execution time* atau waktu yang dihabiskan dalam pencarian rute terpendek. Oleh karena itu, dalam kasus ini, algoritma BFS memiliki waktu yang lebih efisien dalam pencarian rute terpendek dibanding dengan algoritma DFS yang harus melakukan *backtracking*.

BAB V

KESIMPULAN

5.1 Kesimpulan

Algoritma BFS dan DFS adalah algoritma yang efektif dalam membuat Maze Treasure Hunt. BFS lebih efektif dalam mencari jalur terpendek, sedangkan DFS lebih cepat dalam menyelesaikan maze yang lebih kompleks. Selain pada Maze Treasure Hunt, algoritma BFS dan DFS dapat diterapkan pada berbagai jenis masalah yang melibatkan graf, seperti masalah optimisasi, pengaturan jaringan, dan sebagainya.

5.2 Saran

5.3 Komentar dan Refleksi

Pada tugas besar 2 Strategi Algoritma tentang ini, kami belajar tentang algoritma BFS dan DFS dan penerapannya dalam menyelesaikan Maze Treasure Hunt. Selain itu, kami juga belajar bahasa C# untuk membuat GUI Maze Treasure Hunt. Jadi, tugas ini meningkatkan pemahaman kami tentang algoritma BFS dan DFS dan meningkatkan kemampuan pemrograman kami, khususnya pada bahasa C#

DAFTAR PUSTAKA

1. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
2. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
3. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf

LINK REPOSITORY

https://github.com/varrazha/Tubes2STIMA_KepitingMatre

LINK YOUTUBE

https://www.youtube.com/watch?v=1mKfv2NIRRU