

Implementasi Algoritma Depth First Search(DFS) dan Breadth First Search (BFS) dalam Menyelesaikan Persoalan Maze Treasure Hunt

Diajukan sebagai penuhan Tugas Besar 2 Mata Kuliah IF2211 Strategi Algoritma



Oleh:

Kelompok bebas-siiii

1. 13521107 Jericho Russel Sebastian
2. 13521115 Shelma Salsabila
3. 13521131 Jeremya Dharmawan Raharjo

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

DAFTAR ISI	1
BAB I	2
BAB II	4
2.1. Algoritma Program	4
2.1.1 Algoritma Breadth First Search (BFS)	4
2.1.2 Algoritma Depth First Search (DFS)	4
2.1.3 Traveling Salesperson Problem	5
2.2. C# desktop application development	5
2.3 Unity	6
BAB III	7
3.1. Langkah-Langkah Pemecahan Masalah	8
3.1.1 Persoalan Maze Treasure Hunt dengan BFS	8
3.1.2 Persoalan Maze Treasure Hunt dengan DFS	9
3.2. Proses Mapping Persoalan Menjadi Algoritma BFS dan DFS	10
3.2.1 Proses Mapping Persoalan Menjadi Algoritma BFS	11
3.2.2 Proses Mapping Persoalan Menjadi Algoritma DFS	14
3.3. Ilustrasi Kasus Menggunakan Algoritma Lain	16
BAB IV	18
4.1 Implementasi Kode Program	18
4.2 Struktur Data dan Spesifikasi Program	21
4.3 Penjelasan Tata Cara Penggunaan Program	22
4.4 Hasil Pengujian	25
4.5 Analisis Solusi	30
BAB V	36
DAFTAR REFERENSI	37
LAMPIRAN	38

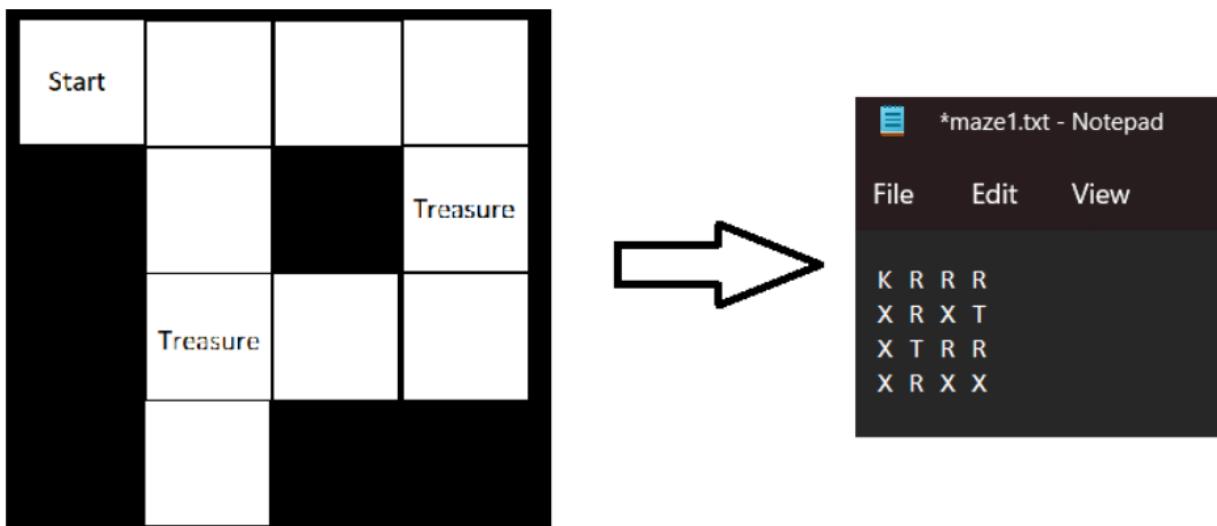
BAB I

DESKRIPSI TUGAS

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

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

Contoh file input:



Gambar 1.1 Ilustrasi input File Maze

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

Input txt tersebut divisualisasikan 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.

BAB II

DASAR TEORI

2.1. Algoritma Program

2.1.1 Algoritma Breadth First Search (BFS)

Algoritma Breadth First Search merupakan algoritma traversal pencarian simpul-simpul pada graf secara melebar. Adapun metode BFS secara umum adalah sebagai berikut:

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

Adapun struktur data pada algoritma ini adalah sebagai berikut:

1. Matriks ketetanggaan $A = [a_{ij}]$ yang berukuran $n \times n$, $a_{ij} = 1$, jika simpul i dan simpul j bertetangga, $a_{ij} = 0$, jika simpul i dan simpul j tidak bertetangga.
2. Antrian q untuk menyimpan simpul yang telah dikunjungi.
3. Tabel Boolean, diberi nama “dikunjungi”
dikunjungi: array[1..n] of boolean
 $dikunjungi[i] = \text{true}$ jika simpul i sudah dikunjungi
 $dikunjungi[i] = \text{false}$ jika simpul i belum dikunjungi

2.1.2 Algoritma Depth First Search (DFS)

Algoritma Depth First Search merupakan algoritma traversal yang mencari simpul-simpul pada graf secara mendalam. Struktur data yang dimiliki kurang lebih sama dengan algoritma Breadth First Search(matriks ketetanggaan, tabel boolean, namun tidak ada struktur data antrian). Adapun metode DFS secara umum secara rekursif adalah sebagai berikut:

1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v.
3. Ulangi DFS mulai dari simpul w.

4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

2.1.3 Traveling Salesperson Problem

Traveling salesperson problem adalah sebuah metode yang digunakan untuk mencari jalan terpendek yang melewati semua kota tepat satu kali dan orang itu diharuskan kembali ke tempat semula. Pada dasarnya persoalan ini adalah menentukan sirkuit hamilton dengan bobot minimum. Biasanya untuk mencari ini digunakan cara exhaustive search. Traveling sales problem ini akan diaplikasikan dalam pengajaran program untuk mengimplementasikan bonus 1. Pada program yang telah dibuat oleh penulis ketika semua treasure telah didapatkan maka program akan berusaha mencari jalan pulang sama halnya seperti pada Travelling sales problem yang mengharuskan penelusuran untuk kembali ke tempat awal.

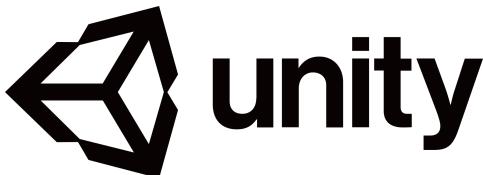
2.2. C# desktop application development

C# Desktop Application Development adalah sebuah kertas pegembang desktop application yang mampu beroperasi tanpa menggunakan internet. Pada umumnya, pengembangan desktop application menggunakan bahasa pemrograman C++, C#, ataupun Java. Pengembangan desktop application, terlebih dalam hal user interface (UI), dipermudah dengan menggunakan WinForms/WPF dan bantuan integrated development environment (IDE) Visual Studio. WPF digunakan untuk pengembangan desktop application berbasis Windows dengan menggunakan bahasa native-nya, yaitu bahasa pemrograman C#. Berikut merupakan langkah-langkah untuk mengembangkan desktop application dengan menggunakan Windows Form App:

1. Membuat Project baru:
 - Buka aplikasi Visual Studio 2022.
 - Pada jendela awal, pilih Create a new project.

- Pada jendela Create a new project, pilih Windows Forms App (.NET Frameworks).
 - Pada jendela Configure your new project, isi nama project pada isian Project name. Kemudian, tekan tombol Create.
2. Membuat aplikasi:
- Pilih menu Toolbox untuk membuka jendela Toolbox fly-out.
 - Pada jendela Toolbox fly-out, kita dapat memilih komponen-komponen yang ingin kita gunakan pada aplikasi desktop kita. Untuk memodifikasi komponen, dapat mengubah properti pada jendela properties.
3. Menambahkan kode pada form:
- Pada jendela Form1.cs [Design], tekan dua kali untuk membuka jendela jendela Form1.cs.
 - Kode dapat ditulis pada file Form1.cs. 4. Menjalankan aplikasi a. Tekan tombol Start pada menu.

2.3 Unity



Gambar 2.1 Unity Game Engine

Unity adalah sebuah game engine (mesin game) yang digunakan untuk membuat berbagai jenis game, baik untuk platform PC, mobile, hingga konsol game. Unity menyediakan berbagai fitur dan tools yang memudahkan developer untuk membuat game dengan cepat dan efisien. Unity mendukung fitur *cross-platform* (unity mendukung berbagai jenis platform seperti Windows, Playstation, Android, dll.) beserta Scripting language, dimana Unity menggunakan bahasa pemrograman C# sebagai bahasa scripting-nya, sehingga developer yang sudah familiar dengan bahasa ini bisa dengan mudah membuat game di Unity. Unity dapat digunakan untuk

membangun aplikasi sederhana berbasis GUI yang sesuai dengan permintaan dari tugas besar ini (beserta bahasa dukungannya, C#).

BAB III

Aplikasi BFS dan DFS Pada Persoalan Maze Treasure Hunt

3.1.Langkah-Langkah Pemecahan Masalah

Untuk merepresentasikan Persoalan Maze Treasure Hunt dengan graf, kita dapat menggunakan struktur data graf yang terdiri dari simpul dan edge. Setiap simpul merepresentasikan ruangan dalam labirin, sementara edge merepresentasikan pintu atau koridor yang menghubungkan ruangan tersebut.Untuk merepresentasikan permainan Labyrinth Treasure Hunt dengan graf, kita dapat menggunakan struktur data graf yang terdiri dari simpul dan edge. Setiap simpul merepresentasikan ruangan dalam labirin, sementara edge merepresentasikan pintu atau koridor yang menghubungkan ruangan tersebut.

3.1.1 Persoalan Maze Treasure Hunt dengan BFS

Adapun langkah-langkah yang bisa diterapkan adalah sebagai berikut.

1. Inisialisasi variabel ‘searchTree’ dengan simpul awal yaitu simpul start.
2. Inisialisasi variabel ‘searchQueue’ dengan simpul awal yaitu simpul start.
3. Inisialisasi objek hash set ‘visited’ untuk menyimpan simpul yang sudah dikunjungi.
4. Pencarian disini akan berhenti ketika menemukan sebuah treasure atau tidak ada jalan lain yang bisa ditempuh.
5. Pencarian awal dilakukan dengan men-check simpul start, kemudian simpul start dimasukkan ke dalam ‘visited’.
6. Semasa proses pencarian, ‘searchQueue’ tidak mungkin kosong.
7. Simpul-simpul yang akan diperiksa disimpan di dalam ‘searchQueue’ kemudian satu-satu di-dequeue. Diperiksa apakah memenuhi syarat pada nomor empat. Jika memenuhi maka proses pencarian selesai dan mengembalikan jalur yang ditemukan.
8. Jika tidak memenuhi kondisi nomor 4 maka, simpul dimasukkan ke dalam ‘visited’ kemudian dicari semua simpul tetangganya.

9. Kemudian simpul tetangga diperiksa apakah sudah dikunjungi dan apakah simpul tetangga bisa dilewati.
10. Jika simpul tetangga belum dikunjungi dan dapat dilewati, tambahkan simpul tetangga ke dalam searchQueue, tandai simpul tetangga sebagai sudah dikunjungi dengan menambahkannya ke dalam objek hash set visited, dan tambahkan simpul tetangga sebagai simpul anak dari simpul saat ini dalam objek searchTree.
11. Ulangi langkah 1 sampai 10 sampai memenuhi syarat poin 4
12. Langkah 1 s/d 11 berfungsi untuk mencari 1 Treasure sedangkan dimungkinkan ada beberapa treasure dalam suatu map dan juga untuk memenuhi salah satu poin bonus program akan dibuat dapat kembali ke tempat awal melaju yaitu dengan cara.
13. Dalam program ini untuk mengetahui jumlah harta karun yang belum didapatkan maka disimpan dalam sebuah variabel ‘treasureTiles’.
14. Program akan mengecek apakah semua harta karun sudah didapatkan jika belum maka akan terus dilakukan pencarian dengan memanggil fungsi awal untuk mencari satu harta karun dengan titik start adalah tempat keberadaan treasure yang terakhir kali ditemukan.
15. Kemudian ada pilihan apakah user ingin kembali ke titik awal atau tidak jika iya maka setelah menemukan semua treasure maka akan dipanggil lagi fungsi pencarian dengan tujuan akhir bukan lagi treasure melainkan posisi start.

3.1.2 Persoalan Maze Treasure Hunt dengan DFS

Adapun langkah-langkah yang bisa dilakukan adalah sebagai berikut.

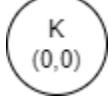
1. Buatlah sebuah pohon pencarian dimulai dari simpul awal start.
2. Buatlah tumpukan stack untuk menyimpan simpul-simpul yang akan dikunjungi.
3. Buatlah sebuah objek hash set untuk menyimpan simpul-simpul yang sudah dikunjungi.
4. Tandai simpul start/awal sebagai sudah dikunjungi masukkan ke hash set.

5. Selama tumpukan masih berisi simpul yang belum dikunjungi, ambil simpul paling atas dari tumpukan (pop), dan periksa apakah simpul tersebut merupakan treasure.
6. Jika simpul tersebut merupakan simpul tujuan, kembalikan jalur yang ditemukan dari simpul awal ke simpul tujuan, dan tandai pencarian telah berhasil (true).
7. Jika tidak, tandai simpul yang diambil sebagai sudah dikunjungi dan tambahkan simpul tersebut sebagai anak dari simpul sebelumnya di pohon pencarian.
8. Periksa setiap simpul tetangga dari simpul yang diambil, dan jika simpul tersebut belum dikunjungi dan dapat dilalui, masukkan simpul tersebut ke dalam stack, dan tandai sebagai sudah dikunjungi.
9. Ulangi langkah 3-7 hingga menemukan treasure atau tidak ada yang bisa diperiksa lagi.
10. Langkah 1 s/d 8 berfungsi untuk mencari 1 Treasure sedangkan dimungkinkan ada beberapa treasure dalam suatu map dan juga untuk memenuhi salah satu poin bonus program akan dibuat dapat kembali ke tempat awal melaju yaitu dengan cara.
11. Dalam program ini untuk mengetahui jumlah harta karun yang belum didapatkan maka disimpan dalam sebuah variabel ‘treasureTiles’.
12. Program akan mengecek apakah semua harta karun sudah didapatkan jika belum maka akan terus dilakukan pencarian dengan memanggil fungsi awal untuk mencari satu harta karun dengan titik start adalah tempat keberadaan treasure yang terakhir kali ditemukan.
13. Kemudian ada pilihan apakah user ingin kembali ke titik awal atau tidak jika iya maka setelah menemukan semua treasure maka akan dipanggil lagi fungsi pencarian dengan tujuan akhir bukan lagi treasure melainkan posisi start.

3.2. Proses Mapping Persoalan Menjadi Algoritma BFS dan DFS

Melakukan mapping persoalan sehingga bisa diselesaikan dengan algoritma BFS dan DFS akan dijelaskan dalam sebuah tabel di bawah ini.

3.2.1 Proses Mapping Persoalan Menjadi Algoritma BFS

Persoalan Mapping Persoalan Menjadi Algoritma BFS		
No	Langkah	Deskripsi
1	<p>Persoalan yang dimiliki misalkan peta yang dimiliki sebagai berikut.</p> <p>K R R R X R X T X T R R X R X X</p> <p>(sumber: spesifikasi tubes)</p>	-
2	Penggambaran tree secara dinamis. Tree yang dibangun dalam program terdapat pada ‘searchTree’	Maka pada proses ini pada awalnya bentuk tree akan terbentuk sebagai berikut. 
3	Cek apakah merupakan treasure. Jika bukan apakah punya jalan lain?	Iya ada jalan lain karena punya tetangga sehingga tree tadi akan berubah statusnya menjadi ‘visited’ dan dilakukan pengecekan terhadap tetangganya. Gambar tree akan menjadi sebagai berikut.

		<p>Ketika sudah menemukan treasure maka program akan berhenti.</p>
4	<p>Jika sudah ditemukan treasure seperti itu maka program akan melanjutkan pencarian treasure yang lain dengan titik awal adalah simpul treasure yang ditemukan paling akhir. Secara berulang dilakukan algoritma ini.</p>	<p>Supaya bisa kembali ke start maka setelah semua treasure ditemukan lakukan pencarian dengan tujuan bukan treasure melainkan titik awal/start.</p>

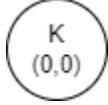
Supaya terlihat perhitungannya, berikut akan dilakukan sebuah perhitungan dalam sebuah notes yang mana pohon yang dibangun akan disusun dengan queue seperti halnya pada program yang telah penulis buat.

Langkah	Isi searchQueue	Enqueue	Dequeue
1	[{0,0}]	{1,0}, {0,1}	-
2	[{1,0}, {0,1}]	-	{0,0}

3	$[\{0,1\}]$	-	$\{1,0\}$ (Tidak ada enqueue pada bagian ini karena karakternya tidak bisa dilewati sehingga tidak mungkin ada hubungan ketetanggan)
4	$[\{0,1\}]$	$\{1,1\}, \{0,2\}$	-
5	$[\{1,1\}, \{0,2\}]$	-	$\{0,1\}$
6	$[\{1,1\}, \{0,2\}]$	$\{2,1\}, \{1,2\}$	-
7	$[\{0,2\}, \{2,1\}, \{1,2\}]$	-	$\{1,1\}$
8	$[\{0,2\}, \{2,1\}, \{1,2\}]$	$\{1,2\}, \{0,3\}$	-
9	$[\{2,1\}, \{1,2\}, \{1,2\}, \{0,3\}]$	-	$\{0,2\}$
10	$[\{2,1\}, \{1,2\}, \{1,2\}, \{0,3\}]$	$\{3,1\}, \{2,2\}$	-
11	$[\{1,2\}, \{1,2\}, \{0,3\}, \{3,1\}, \{2,2\}]$	-	$\{2,1\}$ Pencarian selesai karena treasure sudah ditemukan. Untuk menemukan treasure selanjutnya atau kembali ke awal seperti yang telah dijelaskan sebelumnya maka akan dilakukan

			kembali pencarian dengan titik start adalah tempat treasure yang terakhir kali ditemukan
--	--	--	--

3.2.2 Proses Mapping Persoalan Menjadi Algoritma DFS

Persoalan Mapping Persoalan Menjadi Algoritma DFS		
No	Langkah	Deskripsi
1	<p>Persoalan yang dimiliki misalkan peta yang dimiliki sebagai berikut.</p> <p>K R R R X R X T X T R R X R X X</p> <p>(sumber: spesifikasi tubes)</p>	-
2	<p>Penggambaran tree secara dinamis. Tree yang dibangun dalam program terdapat pada ‘searchTree’</p>	<p>Maka pada proses ini pada awalnya bentuk tree akan terbentuk sebagai berikut.</p> 
3	<p>Cek apakah merupakan treasure. Jika bukan apakah punya jalan lain?</p>	<p>Iya ada jalan lain karena punya tetangga sehingga tree tadi akan berubah statusnya menjadi ‘visited’ dan dilakukan pengecekan terhadap tetangganya. Gambar tree akan menjadi sebagai berikut.</p>

		<p>Ketika sudah menemukan treasure maka program akan berhenti.</p>
4	<p>Jika sudah ditemukan treasure seperti itu maka program akan melanjutkan pencarian treasure yang lain dengan titik awal adalah simpul treasure yang ditemukan paling akhir. Secara berulang dilakukan algoritma ini.</p>	<p>Supaya bisa kembali ke start maka setelah semua treasure ditemukan lakukan pencarian dengan tujuan bukan treasure melainkan titik awal/start.</p>

Supaya terlihat perhitungannya berikut akan dilakukan sebuah perhitungan dalam sebuah notes yang mana pohon yang dibangun akan disusun dengan stack seperti halnya pada program yang telah penulis buat.

Langkah	Isi stack	Push	Pop
1	[{0,0}]	{1,0}, {0,1}	-
2	[{0,1},{1,0}]	-	{0,0}

3	$[\{0,1\}, \{1,0\}]$	$\{0,2\}, \{1,1\}$	-
4	$[\{1,1\}, \{0,2\}, \{1,0\}]$	-	$\{0,1\}$
5	$[\{1,1\}, \{0,2\}, \{1,0\}]$	$\{1,2\}, \{2,1\}$	-
6	$[\{2,1\}, \{1,2\}, \{0,2\}, \{1,0\}]$	-	$\{1,1\}$
7	$[\{2,1\}, \{1,2\}, \{0,2\}, \{1,0\}]$	$\{2,2\}, \{3,1\}$	-
8	$[\{3,1\}, \{2,2\}, \{1,2\}, \{0,2\}, \{1,0\}]$	-	<p>{2,1}</p> <p>Pencarian selesai karena treasure sudah ditemukan. Untuk menemukan treasure selanjutnya atau kembali ke awal seperti yang telah dijelaskan sebelumnya maka akan dilakukan kembali pencarian dengan titik start adalah tempat treasure yang terakhir kali ditemukan</p>

3.3. Ilustrasi Kasus Menggunakan Algoritma Lain

Ada beberapa algoritma lain yang dapat digunakan untuk menyelesaikan maze treasure hunt selain BFS dan DFS, antara lain:

1. A* (A Star): Algoritma pencarian jalur terpendek yang menggunakan heuristik untuk menentukan jalur terbaik. Algoritma ini lebih cepat dan lebih efisien dibandingkan dengan BFS dan DFS, terutama pada kasus-kasus dengan ukuran maze yang besar.

2. Hill Climbing: Algoritma pencarian yang mencoba untuk menemukan solusi terbaik dengan melakukan pergerakan ke arah yang lebih baik secara bertahap, hingga tidak ada lagi langkah yang bisa dilakukan. Meskipun tidak selalu menemukan solusi optimal, hill climbing dapat menemukan solusi yang memadai pada maze yang relatif sederhana.
3. Simulated Annealing: Algoritma pencarian yang didasarkan pada prinsip fisika simulasi proses pemanasan dan pendinginan logam untuk menemukan solusi optimal. Algoritma ini menghasilkan solusi yang baik dengan cara menjelajahi ruang solusi secara acak dengan kemungkinan untuk menerima solusi yang buruk sebelum akhirnya bergerak ke solusi yang semakin baik.

BAB IV

ANALISIS DAN PEMECAHAN MASALAH

4.1 Implementasi Kode Program

Kode kami disusun dalam beberapa file supaya lebih modular. Adapun pseudocode yang kami jelaskan adalah pseudocode mengenai algoritma utama. Ada beberapa yang tidak kami jelaskan seperti tree dan sebagainya yang bertujuan untuk membantu proses pencarian. Ada 3 bagian utama yang akan dijelaskan di pseudocode ini.

1. Pseudocode BFS

Pseudocode BFS
<pre>if(petaAvailable==null)then throw exception if(startPos == null)then throw exception else {Lakukan search Methode} Ignore <- new List<Map<MazeTreasureMap.MazeTileType>.MapTile>() {Memuat elemen-elemen yang harus diabaikan} Search methode: searchTree <- new Tree<start> {Menyimpan start pencarian} searchQueue <- new Queue<Tree<start>> {Menyimpan urutan simpul pencarian} visited = new HashSet<Start> {Menyimpan simpul-simpul yang sudah dikunjungi} I traversal elemenIgnore in Ignore Add elemenIgonore to visited searchRoot = searchTree searchQueue.Enqueue(searchTree) {proses enqueue} visited.Add(searchTree.Value) {simpul yang telah di-enqueue}</pre>

```

diinputkan juga ke visited}

WHILE searchQueue is not empty DO
    searchTree = searchQueue.Dequeue()

{Definisi succesPred adalah ketika bertemu dengan Treasure
atau tidak ada jalan lain}

    if successPred(searchTree.Value) is true then
        path = searchTree.TracePath()

            return MazeTreasureSearchStep with path, true,
searchTree.Value, and searchRoot

        break from loop

    else

        yield MazeTreasureSearchStep with null path, false,
searchTree.Value, and searchRoot

        neighbors = Maze.NeighborsOf(searchTree.Value)
        for elemenneighbor in neighbors do
            if elemenneighbor is not in visited and
MazeTreasureMap.Walkable(neighbor) then
                searchQueue.Enqueue(add elemenneighbor as child to
searchTree)

{walkable artinya simpulnya bisa dilewati}
                visited.Add(elemenneighbor)

    end while

```

2. Pseudocode DFS

Pseudocode DFS

```

if(petaAvailable==null)then
throw exception
if(startPos == null)then
throw exception
else

```

```

{Lakukan search Methode}
Ignore = new List<Map<MazeTreasureMap.MazeTileType>.MapTile>()
{Memuat elemen-elemen yang harus diabaikan}

Search methode:
searchTree <- new Tree<start>      {Menyimpan start pencarian}
searchStack <- new Stack<Tree<start>> {Menyimpan urutan simpul
pencarian namun dalam bentuk stack}
visited = new HashSet<Start>      {Menyimpan simpul-simpul yang
sudah dikunjungi}

I traversal elemenIgnore in Ignore
    Add elemenIgnore to visited
searchRoot = searchTree
searchStack.push(searchTree)      {proses push}
visited.Add(searchTree.Value)      {simpul yang telah di-push
diinputkan juga ke visited}

WHILE searchStack is not empty DO
    searchTree = searchQueue.pop()

{Definisi succesPred adalah ketika bertemu dengan Treasure
atau tidak ada jalan lain}
    if successPred(searchTree.Value) is true then
        path = searchTree.TracePath()
            return MazeTreasureSearchStep with path, true,
searchTree.Value, and searchRoot
        break from loop
    else
        yield MazeTreasureSearchStep with null path, false,
searchTree.Value, and searchRoot
    neighbors = Maze.NeighborsOf(searchTree.Value)
    for elemenneighbor in neighbors do
        if elemenneighbor is not in visited and
MazeTreasureMap.Walkable(neighbor) then
            searchStack.push(add elemenneighbor as child to
)

```

```

searchTree)

{walkable artinya simpulnya bisa dilewati}

    visited.Add(elemenneighbor)

end while

```

3. Pseudocode TSP

Pseudocode TSP

```

{Mengumpulkan simpul-simpul yang ada treasure}
treasureTiles <- new HashSet<MazeTreasureMap.MapTile>()

elemen traversal Maze.Treasures

    treasureTiles.Add(Maze[pos])

{Program utama}

finalPath <- new List<Tree<MazeTreasureMap.MapTile>>()
ignore <- new List<MazeTreasureMap.MapTile>()

elemen traversal Ignore

    Ignore.Add(elemen)

currentStart <- start

while(TreasureCount > 0) do

    {Bisa menggunakan BFS / DFS tinggal panggil fungsinya}

    if(path == null) then {sudah ditemukan treasure}

        {Panggil BFS atau DFS dengan start posisi ditemukan
treasure}

        else {tidak ditemukan treasure}

            Break

    if(ReturnToStart) then

        {Panggil BFS atau DFS dengan definisi succesPred adalah
ketika bertemu dengan start}

```

4.2 Struktur Data dan Spesifikasi Program

Berikut merupakan struktur data beserta spesifikasi program yang digunakan untuk membuat program:

1) Graph

Struktur data ini berfungsi untuk merepresentasikan pohon direktori. Dalam struktur data ini ada node dimana setiap node direpresentasikan dengan *maptile*. Dan gabungan dari mapTile itu adalah sebuah struktur data map.

2) Queue

Queue digunakan untuk menyimpan antrian Node yang akan dikunjungi oleh program saat melakukan traversal dengan metode BFS (Breadth First Search). Berikut merupakan implementasi dari method yang digunakan.

1. Enqueue, berfungsi untuk memasukkan Node ke dalam antrian dengan aturan FIFO atau first in first out.
2. Dequeue, berfungsi untuk mengeluarkan Node dari antrian dengan aturan FIFO atau first in first out.

3) Stack

Stack digunakan untuk menyimpan tumpukan Node (nama folder/direktori) yang akan dikunjungi oleh program saat melakukan traversal dengan metode DFS (Depth First Search). Berikut merupakan implementasi dari method yang digunakan.

1. Push, berfungsi untuk memasukkan Node ke dalam tumpukan dengan aturan LIFO atau last in first out.
2. Pop, berfungsi untuk mengeluarkan Node dari tumpukan dengan aturan LIFO atau last in first out.

4.3 Penjelasan Tata Cara Penggunaan Program

Untuk menjalankan program ini cukup mudah yakni sebagai berikut.

1. Download seluruh program pada tautan
https://github.com/JerichoFletcher/Tubes2_bebas-siiiii kemudian jangan lupa untuk di-extract.
2. Buatlah sebuah file txt dimana di dalamnya terkandung peta yang ingin di-test.
3. Salah satu contoh isi filenya sebagai berikut:

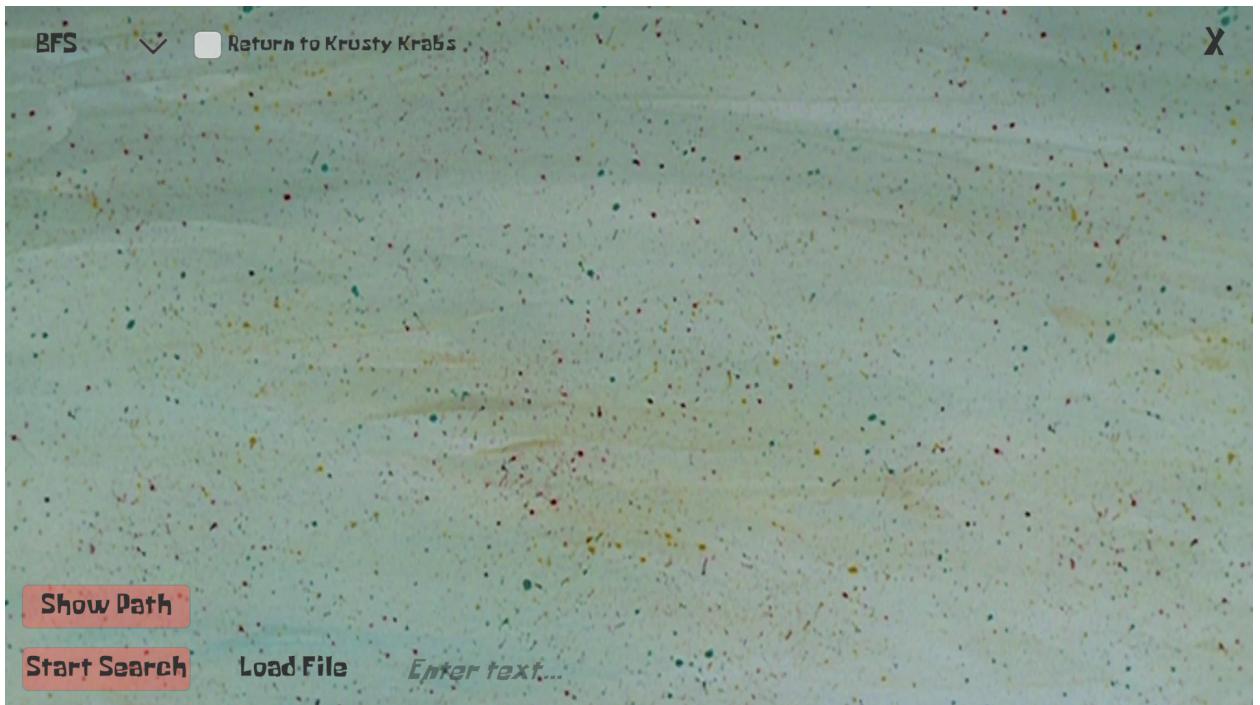
```
X T X X  
X R R T  
K R X T  
X R X R  
X R R R|
```

4. Jangan lupa harus ada karakter K jika tidak maka tidak akan ada penelusuran yang dilakukan serta pastikan isi file hanya mengandung X, T, R, K boleh tidak ada salah satunya.
5. Jika tidak seperti itu misal contohnya seperti di bawah ini:

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

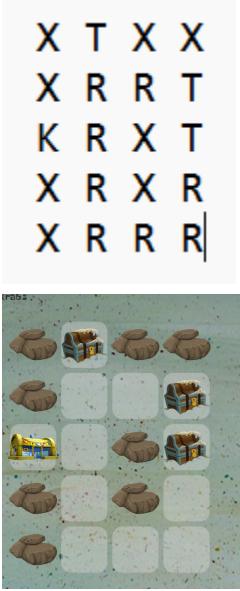
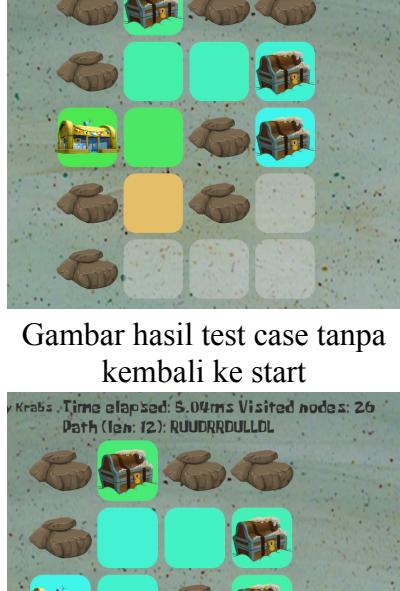
Outputnya akan invalid.

6. Setelah membuat file txt untuk di-test, file ini boleh disimpan di folder test kemudian. jangan lupa copy properties dari file itu/alamat direktori file itu.
7. Setelah di-copy, masuk ke folder bin/GUIApp, kemudian klik BebasFirstVisualize.
8. Setelah itu akan keluar tampilan sebagai berikut.



9. Kemudian inputkan alamat direktori di bagian button yang tertulis ‘Enter text’.
10. Kemudian tekan load file akan muncul hasil konversi dari teks ke map.
11. Kemudian pilih BFS atau DFS jika ingin merubah DFS ke BFS atau sebaliknya klik tombol panah ke arah bawah.
12. Jika ingin proses kembali ke start awal centang ‘Return to Krusty Krab’
13. Kemudian tekan Start Search untuk memulai pencarian

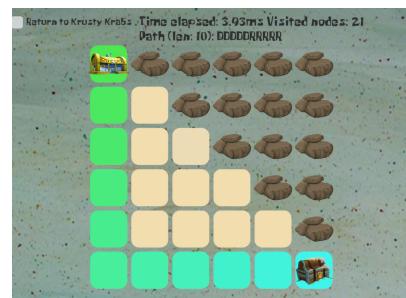
4.4 Hasil Pengujian

No Test Case	Hasil Pengujian dengan BFS	Hasil Pengujian dengan DFS
Test Case 1	<p>X T X X X R R T K R X T X R X R X R R R</p>  <p>Gambar hasil test case tanpa kembali ke start</p> <p>X T X X X R R T K R X T X R X R X R R R</p>  <p>Gambar hasil test case kembali ke start setelah semua treasure ditemukan</p>	 <p>Gambar hasil test case tanpa kembali ke start</p> <p>X T X X X R R T K R X T X R X R X R R R</p>  <p>Gambar hasil test case kembali ke start setelah semua treasure ditemukan</p>

<p>Test Case 2</p> <pre>X T K R T X</pre>	<p>Natura to Krusty Krab . Time elapsed: 19ms Visited nodes: 6 Path (len: 0): LNR</p>	<p>Natura to Krusty Krab . Time elapsed: 135ms Visited nodes: 7 Path (len: 5): RKLRL</p>
	<p>Gambar hasil test case tanpa kembali ke start</p> <p>Natura to Krusty Krab . Time elapsed: 19ms Visited nodes: 9 Path (len: 6): LURRL</p>	<p>Gambar hasil test case tanpa kembali ke start</p> <p>Natura to Krusty Krab . Time elapsed: 145ms Visited nodes: 9 Path (len: 6): RULLR</p>
<p>Test Case 3</p> <pre>J A N G A N L U P A C E K Y A N G B E G I N I Y</pre>	<p>Return to Krusty Krab . File is invalid!</p>	<p>Return to Krusty Krab . File is invalid!</p>

Test Case 4

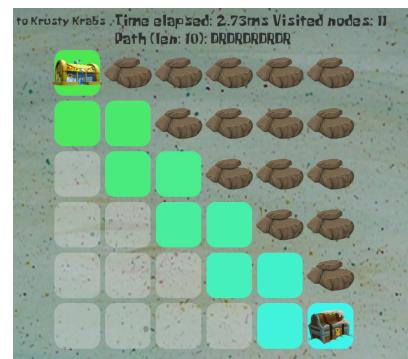
```
K X X X X X X  
R R X X X X X  
R R R X X X  
R R R R X X  
R R R R R X  
R R R R T
```



Gambar hasil test case tanpa kembali ke start



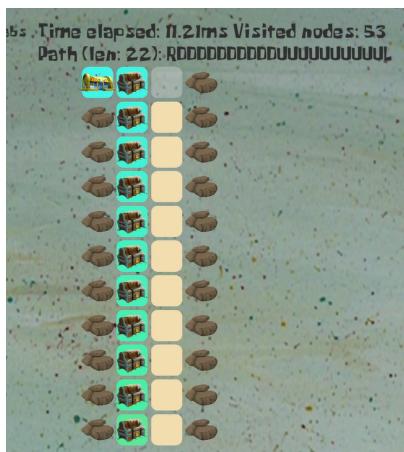
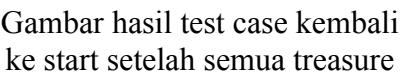
Gambar hasil test case kembali ke start setelah semua treasure ditemukan



Gambar hasil test case tanpa kembali ke start



Gambar hasil test case kembali ke start setelah semua treasure ditemukan

<p>Test Case 5</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>K</td><td>T</td><td>R</td><td>X</td></tr> <tr><td>X</td><td>T</td><td>R</td><td>X</td></tr> </table>	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	X	T	R	X	X	T	R	X	 <p>Krabbs. Time elapsed: 11.21ms Visited nodes: 53 Path (len: 22): RDDDDDDDDDDDDUUUUUUUUUU</p>	 <p>. Time elapsed: 20.26ms Visited nodes: 101 Path (len: 49): RRDDBDDBDRDDDRDLDLRRUULURDDLRUULURUULRUL</p>
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																																																			
X	T	R	X																																																			
X	T	R	X																																																			
	<p>Gambar hasil test case tanpa kembali ke start</p>	<p>Gambar hasil test case tanpa kembali ke start</p>																																																				
	 <p>Gambar hasil test case kembali ke start setelah semua treasure ditemukan</p>	 <p>Gambar hasil test case kembali ke start setelah semua treasure ditemukan</p>																																																				

Adapun secara lebih jelas untuk melihat langkah per langkah dari algoritma berjalan dapat dilihat pada tautan - tautan berikut.

Percobaan	Link Tautan
Test Case 1 BFS	https://drive.google.com/file/d/1WGzwKwYJBOTuc3JMeR1ZuJU392BNnXQE/view?usp=sharing
Test Case 1 DFS	https://drive.google.com/file/d/17-ALXDDefk

	5l2hR6RyxTuD0Xz55_Ynzn/view?usp=sharing
Test Case 2 BFS	https://drive.google.com/file/d/1VTBX7FfpgeIkYGkFx10WCN_YmS6MNEY4/view?usp=sharing
Test Case 2 DFS	https://drive.google.com/file/d/1jfr-NkYjm0_ZG0otcxXsNDmGrEbfuizb/view?usp=sharing
Test Case 3 BFS	Tidak ada jalur yang ditempuh karena file invalid
Test Case 3 DFS	Tidak ada jalur yang ditempuh karena file invalid
Test Case 4 BFS	https://drive.google.com/file/d/1uLS3CU90fqElC9q-95XvP8CoTlapG07/view?usp=sharing
Test Case 4 DFS	https://drive.google.com/file/d/1grugB4PzCXDJnzIOaDdWTrbwn-mZq875/view?usp=sharing
Test Case 5 BFS	https://drive.google.com/file/d/13NR_CDuZzW9t51U2h_ISYA3BiQw8Tbgv/view?usp=sharing
Test Case 5 DFS	https://drive.google.com/file/d/18RyaAOLgOXKLzPXnGP8Z9ba5UXBnXmgK/view?usp=sharing

Beberapa informasi tambahan yang harus dipahami terkait dengan proses test case ini.

1. Karakter start yaitu Krusty Krab digambarkan dengan krusty krab
2. Treasure digambarkan dengan harta karun
3. Jalan yang bisa dilalui atau karakter ‘R’ digambarkan dengan cell warna bening
4. Jalan yang tidak bisa dilalui digambarkan dengan karakter batu
5. Adapun semakin sering suatu cell dilalui warnanya menjadi semakin tua.

4.5 Analisis Solusi

Untuk menganalisis solusi mana yang lebih baik diantara DFS dan BFS akan dilakukan sebuah analisis dari kelima test case yang telah dilakukan di atas.

- Test Case 1

BFS	
Tidak Kembali ke Krusty Krab	
Waktu (ms)	3.11
Jumlah node yang dikunjungi	14
Panjang path	7
Kembali ke Krusty Krab	
Waktu (ms)	5.04
Jumlah node yang dikunjungi	26
Panjang path	12

DFS	
Tidak Kembali ke Krusty Krab	
Waktu (ms)	6.17
Jumlah node yang dikunjungi	23
Panjang path	18
Kembali ke Krusty Krab	
Waktu (ms)	9.55
Jumlah node yang dikunjungi	28
Panjang path	22

Kesimpulannya: Dari hasil tersebut penulis akan menyimpulkan mana yang lebih baik ketika waktu yang dibutuhkan lebih kecil. Namun, terkadang waktu yang dihasilkan dari beberapa percobaan yang telah penulis lakukan kurang valid sehingga akan dibantu dengan membandingkan jumlah node yang dikunjungi. Dari percobaan di atas:

1. Ketika tidak kembali ke krusty crab waktu BFS node yang dikunjungi juga lebih sedikit sehingga dari sini dapat dilihat BFS lebih baik dari DFS
2. Ketika kembali ke krusty crab waktu BFS lebih cepat dan jumlah node yang dikunjungi juga lebih sedikit sehingga pada hal ini BFS lebih unggul.

Dari percobaan ini BFS memiliki dua poin unggul dibanding DFS.

- Test Case 2

BFS	
Tidak Kembali ke Krusty Krab	
Waktu (ms)	1.41
Jumlah node yang dikunjungi	6
Panjang path	4
Kembali ke Krusty Krab	
Waktu (ms)	1.89
Jumlah node yang dikunjungi	9
Panjang path	6

DFS	
Tidak Kembali ke Krusty Krab	
Waktu (ms)	1.55
Jumlah node yang dikunjungi	7
Panjang path	5

Kembali ke Krusty Krab	
Waktu (ms)	1.45
Jumlah node yang dikunjungi	9
Panjang path	6

Kesimpulannya: Dari hasil tersebut penulis akan menyimpulkan mana yang lebih baik ketika waktu yang dibutuhkan lebih kecil. Namun, terkadang waktu yang dihasilkan dari beberapa percobaan yang telah penulis lakukan kurang valid sehingga akan dibantu dengan membandingkan jumlah node yang dikunjungi. Dari percobaan di atas:

1. Ketika tidak kembali ke krusty krab waktu BFS lebih cepat node yang dikunjungi lebih sedikit dibanding DFS sehingga BFS lebih unggul.
 2. Ketika kembali ke krusty krab waktu DFS lebih cepat dan jumlah node yang dikunjungi juga tidak lebih banyak dari BFS sehingga pada hal ini DFS lebih unggul.
- Dari percobaan ini DFS memiliki satu poin dan BFS memiliki satu poin.

- Test Case 3

BFS	
Tidak Kembali ke Krusty Krab	
Waktu (ms)	-
Jumlah node yang dikunjungi	-
Panjang path	-
Kembali ke Krusty Krab	
Waktu (ms)	-
Jumlah node yang dikunjungi	-
Panjang path	-

DFS	
Tidak Kembali ke Krusty Krab	
Waktu (ms)	-
Jumlah node yang dikunjungi	-
Panjang path	-
Kembali ke Krusty Krab	
Waktu (ms)	-
Jumlah node yang dikunjungi	-
Panjang path	-

Kesimpulannya: Tidak ada kesimpulan yang dihasilkan karena file map invalid.

- Test Case 4

BFS	
Tidak Kembali ke Krusty Krab	
Waktu (ms)	3.93
Jumlah node yang dikunjungi	21
Panjang path	10
Kembali ke Krusty Krab	
Waktu (ms)	7.55
Jumlah node yang dikunjungi	42
Panjang path	20

DFS	
Tidak Kembali ke Krusty Krab	

Waktu (ms)	2.73
Jumlah node yang dikunjungi	11
Panjang path	10
Kembali ke Krusty Krab	
Waktu (ms)	5.67
Jumlah node yang dikunjungi	27
Panjang path	22

Kesimpulannya: Dari hasil tersebut penulis akan menyimpulkan mana yang lebih baik ketika waktu yang dibutuhkan lebih kecil. Namun, terkadang waktu yang dihasilkan dari beberapa percobaan yang telah penulis lakukan kurang valid sehingga akan dibantu dengan membandingkan jumlah node yang dikunjungi. Dari percobaan di atas:

1. Ketika tidak kembali ke krusty krab waktu DFS lebih cepat node yang dikunjungi juga jauh lebih sedikit sehingga pada hal ini DFS lebih unggul dari BFS.
 2. Ketika kembali ke krusty krab hasilnya juga sama seperti kasus no 1
Dari percobaan ini DFS memiliki dua poin unggul dibanding BFS.
- Test Case 5

BFS	
Tidak Kembali ke Krusty Krab	
Waktu (ms)	6.53
Jumlah node yang dikunjungi	31
Panjang path	11
Kembali ke Krusty Krab	
Waktu (ms)	11.21
Jumlah node yang dikunjungi	53

Panjang path	22
--------------	----

DFS	
Tidak Kembali ke Krusty Krab	
Waktu (ms)	26.56
Jumlah node yang dikunjungi	78
Panjang path	44
Kembali ke Krusty Krab	
Waktu (ms)	20.26
Jumlah node yang dikunjungi	101
Panjang path	46

Kesimpulannya: Dari hasil tersebut penulis akan menyimpulkan mana yang lebih baik ketika waktu yang dibutuhkan lebih kecil. Namun, terkadang waktu yang dihasilkan dari beberapa percobaan yang telah penulis lakukan kurang valid sehingga akan dibantu dengan membandingkan jumlah node yang dikunjungi. Dari percobaan di atas:

1. Ketika tidak kembali ke krusty krab waktu BFS lebih cepat node yang dikunjungi juga jauh lebih sedikit sehingga pada hal ini BFS lebih unggul dari DFS.
2. Ketika kembali ke krusty krab waktu BFS lebih cepat dan jumlah node yang dikunjungi juga lebih kecil dari DFS sehingga dapat dikatakan BFS unggul dibandingkan DFS.

Dari percobaan ini BFS memiliki dua poin unggul dibanding DFS.

Sehingga dari hasil analisis itu BFS memiliki 5 poin sedangkan DFS memiliki 3 poin dapat disimpulkan dari test case yang ada algoritma pencarian treasure dengan BFS jauh lebih cepat. Hanya saja perbedaannya cukup sedikit tidak bisa disimpulkan secara mutlak seperti itu karena disesuaikan dengan bagaimana bentuk mazanya.

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

Dari Tugas Besar I IF2211 Strategi Algoritma Semester II 2022/2023 berjudul *Implementasi Algoritma Depth First Search (DFS) dan Breadth First Search (BFS) dalam Menyelesaikan Persoalan Maze Treasure Hunt*, kami mendapati bahwa untuk melakukan pencarian rute dapat menerapkan konsep pencarian pada graf menggunakan algoritma Breadth First Search dan Depth First Search.

Tugas besar ini memaparkan dan mengenalkan kami secara lanjut terhadap bahasa C# sebagai salah satu bahasa berorientasi objek yang dapat digunakan untuk mengembangkan aplikasi pada desktop dengan Visual Studio. Kami juga tak luput mengeksplorasi penggunaan Unity dari sisi tampilan untuk lebih baik memvisualisasikan apa yang telah kami buat.

Algoritma Breadth First Search dan Depth First Search memiliki performa yang berbeda untuk tiap-tiap kasus Maze Treasure Hunt. Secara heuristik, algoritma Breadth First Search memiliki performa yang lebih baik untuk pencarian dengan banyak simpul graf yang bercabang, sedangkan algoritma Depth First Search lebih baik untuk pencarian dengan simpul graf yang memiliki sedikit cabang dan dibutuhkan pencarian yang mendalam untuk mengeksplorasi solusi.

Kami menyadari tugas besar yang kami buat tak luput dari kekurangan. Maka dari itu, kami masih perlu banyak belajar, baik dari sisi algoritma yang kami buat untuk lebih adaptif, maupun dari sisi rekayasa perangkat lunak yang memanfaatkan bahasa-bahasa dan paradigma baru.

DAFTAR REFERENSI

Munir, Rinaldi, Nur Ulfa Mauladevi. 2021, “Diktat Breadth/Depth First Search” <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>, diakses pada 18 Maret 2023

Microsoft. “C# Documentation”, <https://docs.microsoft.com/en-us/dotnet/csharp/>, diakses pada 18 Maret 2023

LAMPIRAN

Tautan repository tugas besar:

https://github.com/JerichoFletcher/Tubes2_bebas-siiiii

Tautan Video:

<https://youtu.be/kgCz3iS5AkE>