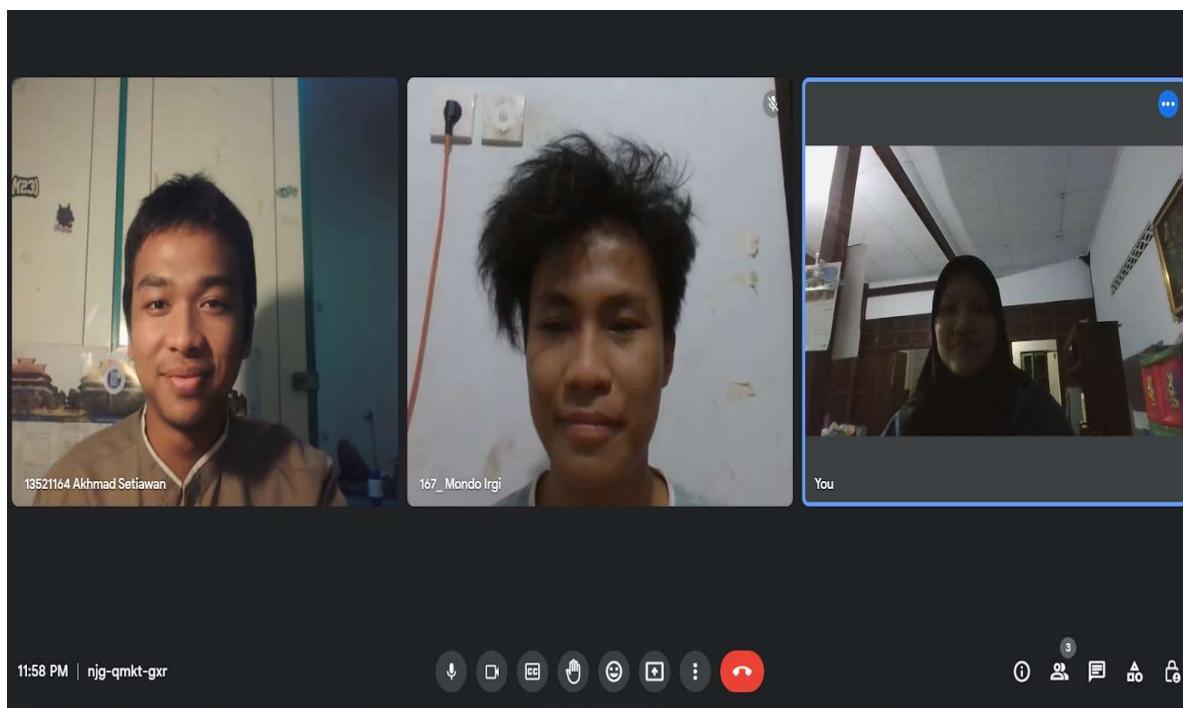


Tugas Besar 1 IF2211 Strategi Algoritma

Semester II Tahun 2022/2023

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



Disusun oleh:

Ferindya Aulia Berlianty 13521161

Akhmad Setiawan 13521164

Irgiansyah Mondo 13521167

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

DAFTAR ISI

DAFTAR ISI

BAB I DESKRIPSI TUGAS

BAB II LANDASAN TEORI

- 2.1 Graph Traversal
- 2.2 Breadth-First Search (BFS)
- 2.3 Depth-First Search (DFS)
- 2.4 Bahasa Pemrograman C#
- 2.5 Penjelasan Mengenai C# Desktop Application Development

BAB III ANALISIS PEMECAHAN MASALAH

- 3.1 Langkah-Langkah Pemecahan Masalah
- 3.2 Proses Mapping Persoalan Menjadi Elemen-Elemen Algoritma BFS dan DFS
- 3.3 Contoh Ilustrasi Kasus Lain

BAB IV IMPLEMENTASI DAN PENGUJIAN

- 4.1 Implementasi Program
- 4.2 Struktur Data
- 4.3 Tata Cara Penggunaan Program
- 4.4 Hasil Pengujian
- 4.5 Analisis Desain Solusi Algoritma BFS dan DFS berdasarkan Hasil Pengujian

BAB V KESIMPULAN DAN SARAN

- 5.1 Kesimpulan
- 5.2 Saran
- 5.3 Refleksi
- 5.4 Komentar

DAFTAR PUSTAKA

LAMPIRAN

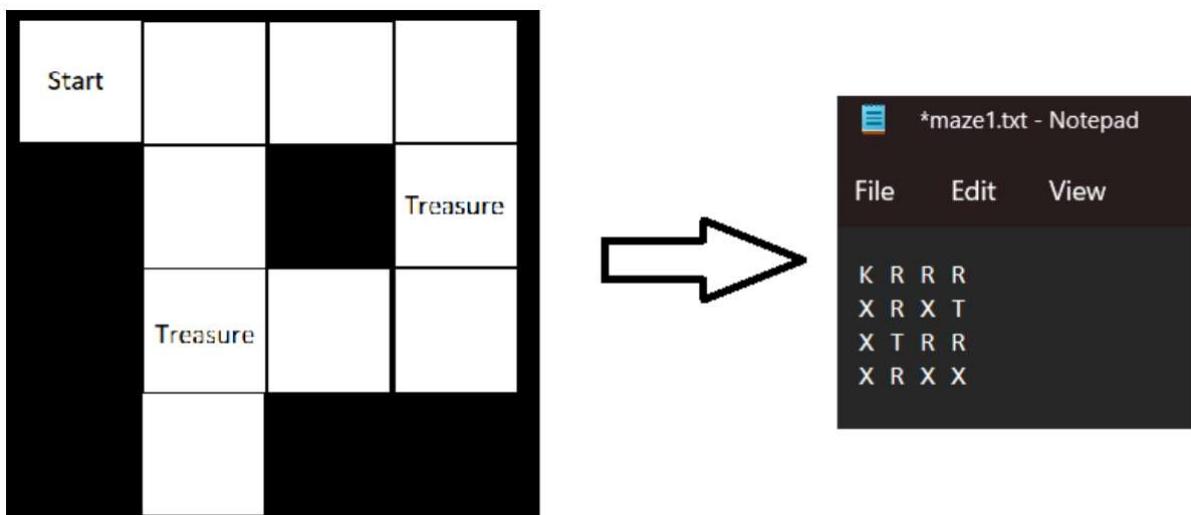
BAB I

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

Contoh file input:

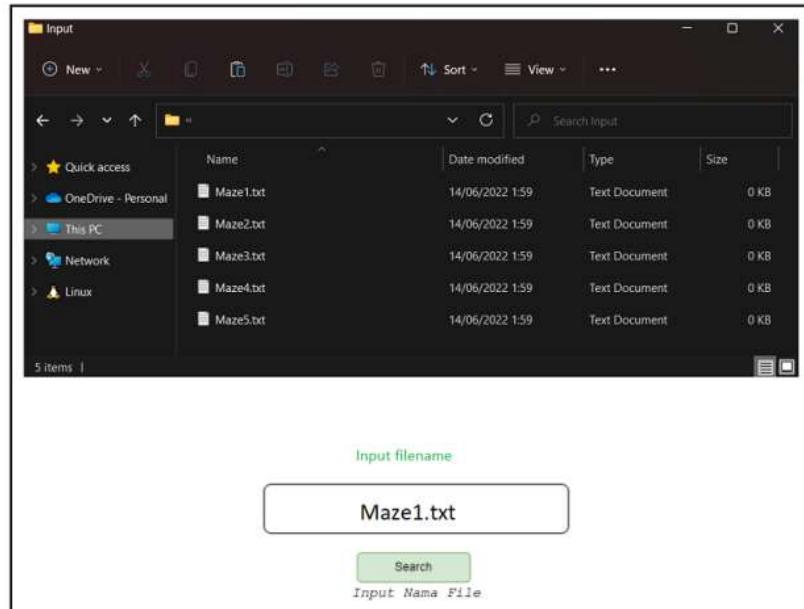


Gambar 1.1 Ilustrasi input file maze

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.

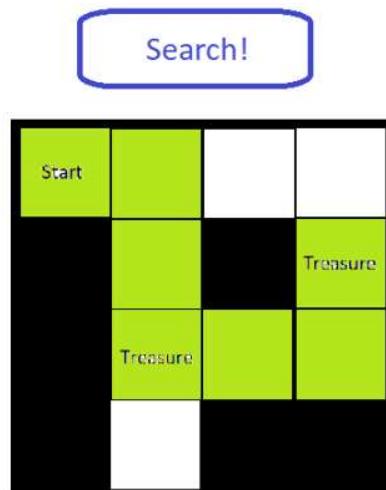
Contoh input aplikasi:



Gambar 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 menghandle kasus apabila tidak ditemukan dengan nama file tersebut.

Contoh output aplikasi:

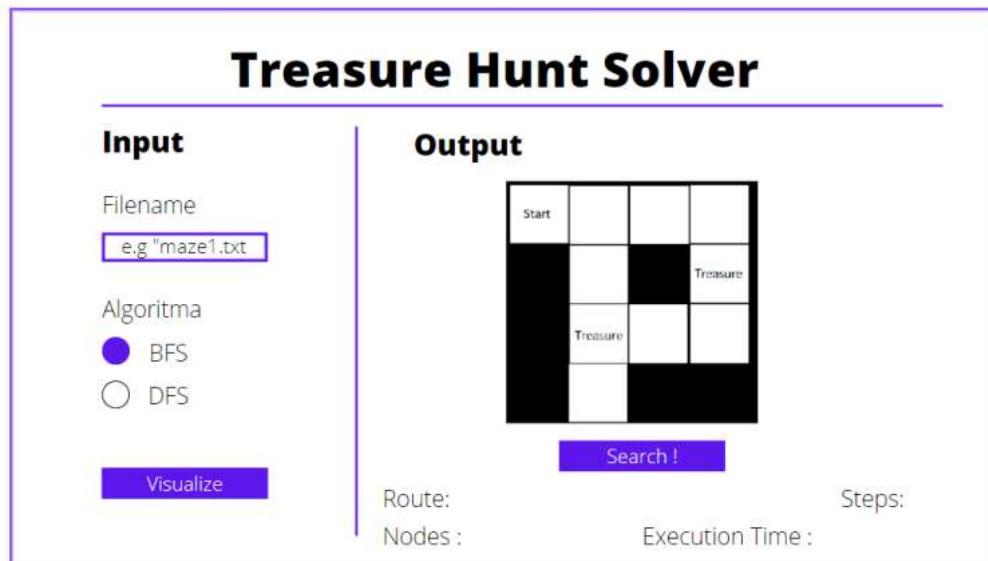


Gambar 1.3 Contoh output program untuk gambar 1.1

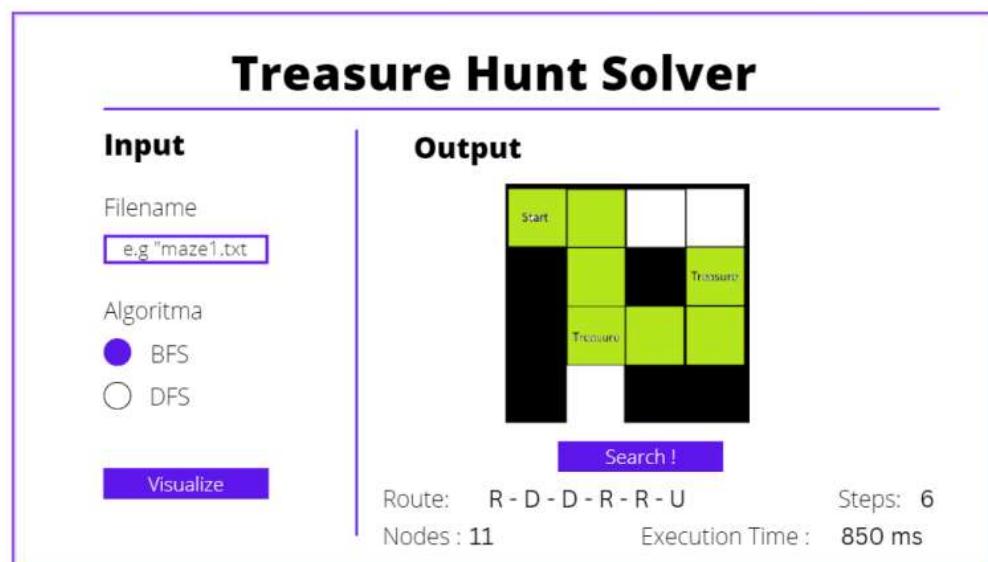
Setelah program melakukan pembacaan input, program akan memvisualisasikan gridnya terlebih dahulu tanpa pemberian rute solusi. Hal tersebut dilakukan agar pengguna dapat mengerjakan terlebih dahulu treasure hunt secara manual jika diinginkan. Kemudian, program menyediakan tombol solve untuk mengeksekusi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

Spesifikasi Program:

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



Gambar 1.4 Tampilan Program Sebelum dicari solusinya



Gambar 1.5 Tampilan Program setelah dicari solusinya

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.

Spesifikasi GUI:

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

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.

Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk memvisualisasikan matrix dalam bentuk grid adalah DataGridView. Berikut adalah panduan singkat terkait penggunaannya <http://csharp.net-informations.com/datagridview/csharp-datagridview-tutorial.htm>

7. Mahasiswa tidak diperkenankan untuk melihat atau menyalin library lain yang mungkin tersedia bebas terkait dengan pemanfaatan BFS dan DFS. Akan tetapi, untuk algoritma lain diperbolehkan menggunakan library jika ada.

BAB II

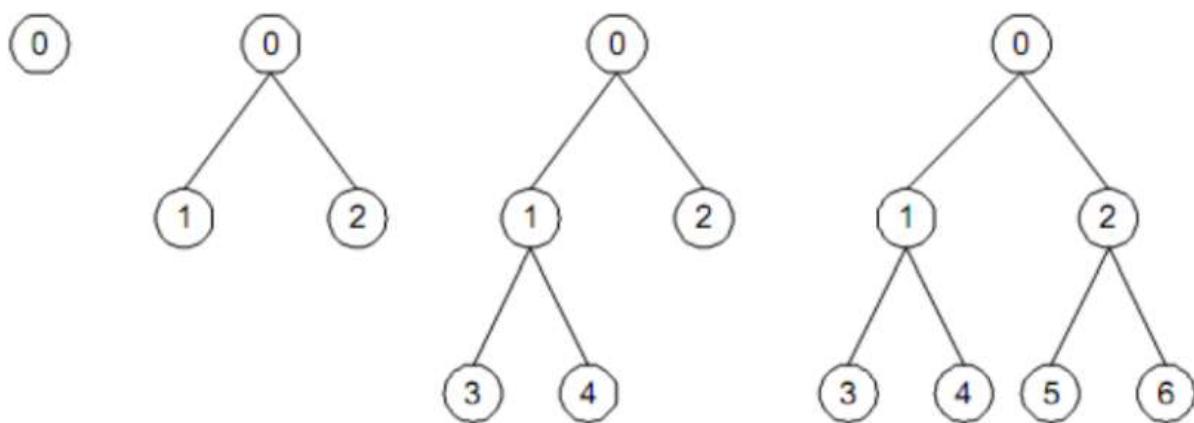
LANDASAN TEORI

2.1 Graph Traversal

Graph Traversal merupakan salah satu algoritma penelusuran pada graph yang dilakukan dengan cara mengunjungi setiap simpul atau node yang ada pada graph secara sistematik. Graph Traversal terdiri dari 2 metode yaitu *Breadth First Search (BFS)* dan *Depth First Search (DFS)*. Terdapat dua pendekatan representasi graph dalam proses pencarian yaitu graf statis dan graf dinamis. Graf statis merupakan graf yang sudah terbentuk sebelum proses pencarian dilakukan, sedangkan graf dinamis merupakan graf yang terbentuk saat proses pencarian dilakukan.

2.2 Breadth-First Search (BFS)

Breadth-First Search (BFS) atau Pencarian Melebar merupakan algoritma graf traversal dengan mengunjungi sebuah simpul, kemudian melanjutkan kunjungan terhadap seluruh simpul yang bertetangga dengan simpul tersebut. Setelah dilakukan pemeriksaan terhadap seluruh simpul tetangga, pencarian dilanjutkan terhadap simpul tetangga tersebut yang belum dikunjungi, hal ini dilakukan secara terus menerus. *Breadth-First Search (BFS)* merupakan salah satu algoritma yang paling sederhana dalam melakukan pencarian pada graf dan pola dasar pada algoritma pencarian graf lainnya. Algoritma *minimum-spanning tree* Prim dan Algoritma pencarian jarak terdekat Dijkstra menggunakan ide yang serupa dengan pencarian *breadth-first search*. Dalam algoritma *breadth-first search*, graf direpresentasikan dalam bentuk $G = (V, E)$ dengan v merupakan simpul awal penelusuran. Ilustrasi pembentukan pohon ruang status pada BFS dapat dilihat pada gambar berikut.



Gambar 2.1 Contoh Pembentukan Pohon Ruang Status pada BFS

Pada prosedur breadth-first search, dapat menggunakan adjacency lists dalam merepresentasikan graf $G = (V, E)$. Selain itu, untuk mengetahui simpul yang akan diperiksa, akan digunakan struktur data Queue. Terakhir, untuk mengetahui suatu simpul telah diperiksa atau tidak, akan digunakan struktur data *array* atau *hash table* yang bertipe boolean. Berikut adalah *pseudocode* umum untuk prosedur *breadth-first search*.

```

procedure BFS(input G: graph, input v: vertice)
{ Traversal Graf dengan algoritma penelusuran BFS
  I.S. G dan v terdefinisi dan tidak sembarang
  F.S. Semua simpul yang dilalui tercetak pada layar }

KAMUS
{ Variabel }
q : queue
w : vertice
visited : hashTable
{ Fungsi dan Prosedur antara }

procedure createQueue(input/output q: queue)
{ Inisialisasi queue
  I.S. q belum terdefinisi
  F.S. q terdefinisi dan kosong }

procedure enqueue(input/output q: queue, input v: vertice)
{ Memasukkan v ke dalam q dengan aturan FIFO
  I.S. q terdefinisi
  F.S. v masuk ke dalam q dengan aturan FIFO }

function dequeue(q: queue) → vertice
{ Menghapus simpul dari q dengan aturan FIFO
  dan mengembalikan simpul yang dihapus}

function isQueueEmpty(q: queue) → boolean
{ Mengembalikan True jika q kosong dan sebaliknya }

procedure createHashTable(input/output T: hashTable)
{ Inisialisasi hashTable
  I.S. T belum terdefinisi
  F.S. T terdefinisi dan terisi False sebanyak simpul }

procedure setHashTable(input/output T: hashTable, input v: vertice, input val: boolean)
{ Mengubah value dari T
  I.S. T sudah terdefinisi
  F.S. Elemen T dengan key v sudah diubah menjadi val }

function getHashTable(T: hashTable, input v: vertice) → boolean
{ Mengembalikan elemen T pada key v }

ALGORITMA
{ Inisialisasi visited }
createHashTable(visited)
{ Inisialisasi q }
createQueue(q)

```

```

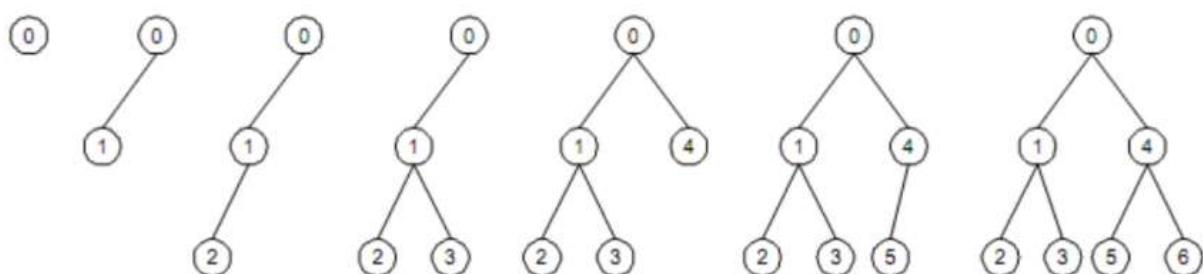
{ Mengunjungi simpul pertama (akar) }
output(v)
enqueue(q, v)
setHashTable(T, v, True)

{ Mengunjungi semua simpul }
while (not isEmpty(q)) do
    v ← dequeue(q)
    for setiap simpul w yang bertetangga dengan simpul v do
        if (not getHashTable(visited, w)) then
            output(w)
            enqueue(q, w)
            setHashTable(T, w, True)
{ q kosong }

```

2.3 Depth-First Search (DFS)

Depth-First Search (DFS) atau biasa disebut dengan pencarian secara mendalam, merupakan algoritma graf traversal yang dilakukan dengan cara mengunjungi sebuah simpul, kemudian melanjutkan kunjungan terhadap sebuah simpul yang bertetangga dengan simpul tersebut, demikian seterusnya. Setelah mencapai suatu simpul sedemikian sehingga seluruh simpul yang bertetangga dengannya telah dikunjungi, pencarian runut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul tetangga yang belum dikunjungi. Pencarian berakhir ketika tidak ada simpul yang belum atau dapat dikunjungi dari simpul yang telah dikunjungi. Ilustrasi pembentukan pohon ruang status pada DFS dapat dilihat pada gambar berikut.



Gambar 2.2 Contoh Pembentukan Pohon Ruang Status pada DFS

Pada prosedur *depth-first search*, representasi graf dan struktur data hampir serupa seperti yang digunakan pada prosedur *breadth-first search*. Akan tetapi, pada *depth-first search*, tidak akan digunakan struktur data queue, melainkan lebih mirip dengan struktur data stack. Berikut merupakan pseudocode dari prosedur *depth-first search*.

```

procedure DFS(input G: graph, input/output visited: hashTable, input v: vertice)
{ Traversal Graf dengan algoritma penelusuran DFS
  I.S. G, v, dan visited terdefinisi dan tidak sembarang
  F.S. Semua simpul yang dilalui tercetak pada layar }

KAMUS
{ Variabel }
w : vertice
{ Fungsi dan Prosedur antara }

procedure setHashTable(input/output T: hashTable, input v: vertice, input val: boolean)
{ Mengubah value dari T
  I.S. T sudah terdefinisi
  F.S. Elemen T dengan key v sudah diubah menjadi val }

function getHashTable(T: hashTable, input v: vertice) → boolean
{ Mengembalikan elemen T pada key v }

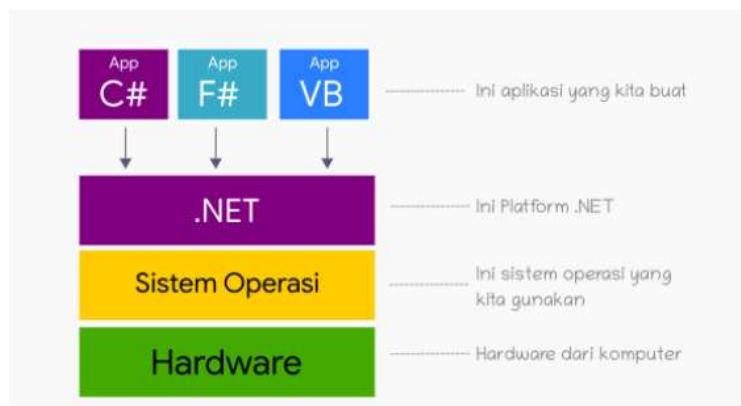
ALGORITMA
{ Mengunjungi simpul sekarang }
output(v)
setHashTable(T, v, True)

{ Mengunjungi semua simpul }
for setiap simpul w yang bertetangga dengan simpul v do
  if (not getHashTable(visited, w)) then
    DFS(G, visited, w)

```

2.4 Bahasa Pemrograman C#

C# merupakan bahasa pemrograman yang dibuat oleh Microsoft dan ditargetkan berjalan di atas platform .NET. .NET merupakan mesin virtual yang digunakan untuk menjalankan program C#, F#, VB.NET dan program lainnya. Selain itu, .NET juga menyediakan tools, library, dan API yang dibutuhkan untuk membuat bahasa pemrograman C#. Ilustrasi hubungan antara C# dan .NET dapat dilihat pada ilustrasi berikut.



Gambar 2.3 Ilustrasi Hubungan antara C# dan .NET

Pada umumnya, program C# akan di-compile menjadi CIL (Common Intermediate Language) yang dipahami oleh .NET sehingga bahasa pemrograman C# cukup berbeda dengan bahasa pemrograman C dan C++ yang di-compile menjadi bahasa assembly dan dapat langsung dieksekusi oleh processor.

2.5 Penjelasan Mengenai C# Desktop Application Development

C# Desktop Application Development adalah sebuah kelas pengembang 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), diperlakukan 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#.

Visual Studio menyediakan berbagai jenis project template dalam berbagai bahasa pemrograman, platform, dan tipe. Untuk pengembangan Desktop Application dengan bahasa pemrograman C#, Visual Studio menyediakan berbagai template yang dapat dimanfaatkan pembuat program untuk membangun aplikasinya. Dalam Tugas Besar II IF2211 Strategi Algoritma ini, kelompok kami menggunakan “Windows Forms App (.NET Framework) sebagai *project template* awal kami.

Berikut merupakan langkah-langkah untuk mengembangkan *desktop application* dengan menggunakan Windows Form App:

1. Membuat *Project* baru:
 - a. Buka aplikasi Visual Studio 2022.
 - b. Pada jendela awal, pilih *Create a new project*.
 - c. Pada jendela *Create a new project*, pilih *Windows Forms App (.NET Framework)*.
 - d. Pada jendela *Configure your new project*, isi nama project pada isian *Project name*. Kemudian, tekan tombol *Create*.
2. Membuat aplikasi:
 - a. Pilih menu *Toolbox* untuk membuka jendela *Toolbox fly-out*.
 - b. 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:

- a. Pada jendela Form1.cs [Design], tekan dua kali untuk membuka jendela jendela Form1.cs.
 - b. Kode dapat ditulis pada file Form1.cs.
4. Menjalankan aplikasi
 - a. Tekan tombol *Start* pada *menu*.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Dalam proses pemecahan permasalahan pengaplikasian algoritma BFS dan DFS dalam menyelesaikan persoalan *Maze Treasure Hunt* ini, kelompok kami melakukan langkah-langkah sebagai berikut. Pertama-tama, kelompok kami memahami terlebih dahulu permasalahan yang tertera pada Bab I Deskripsi Tugas. Langkah berikutnya, kami mempelajari terlebih dahulu konsep, teknik, dan kakas yang dibutuhkan dalam implementasi program, seperti mempelajari konsep pemrosesan folder dan file, pengimplementasian algoritma BFS dan DFS dalam bahasa pemrograman C#, penggunaan kakas Visual Studio dan framework .NET, dan visualisasi *maze*.

3.2 Proses Mapping Persoalan Menjadi Elemen-Elemen Algoritma BFS dan DFS

Elemen-elemen algoritma BFS dan DFS dipetakan dari permasalahan menjadi seperti berikut ini:

1. Algoritma BFS dan DFS merupakan algoritma pemrosesan struktur data graf sehingga dibentuk 3 buah larik, antara lain
 - a. Larik yang mencatat seluruh nodes
 - b. Larik yang mencatat seluruh parent nodes
 - c. Larik yang mencatat seluruh child nodes
2. Pada algoritma DFS, pencarian dilakukan secara rekursif sehingga akan muncul elemen pencatat rute pencarian saat rekurens dieksekusi.
3. Pada algoritma BFS, pencarian dilakukan secara berurutan berdasarkan kunjungan ke semua child terlebih dahulu sehingga digunakan queue untuk mengatur nodes prioritas yang perlu dieksekusi dahulu.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program Utama (BFS dan DFS Solver)

4.1.1 BFSSolver

```
procedure BFSSolver(input: dgv, pathTracker):
{I.S : menerima input datagrid dan tracker untuk path
F.S : datagrid diwarnai sesuai aturan BFS dan pathTracker mencatat path yang dilalui}

    for i in 0 to numRows:
        for j in 0 to numCols:
            if path[i][j] != "X":
                indexing = i * numCols + j
                init = []
                tracker.add(indexing, init)

                queueToVisit.enqueue(startIndex)
                dgv[startIndex[1], startIndex[0]].Style.BackColor =
GetColor(adj[startIndex[0]][startIndex[1]])
                if stepByStep is true:
                    dgv.Refresh()
                    Thread.Sleep(700)
                adj[startIndex[0]][startIndex[1]]++

                currentIndex = startIndex
                while queueToVisit is not empty and treasureCount is not 0:
                    currentIndex = queueToVisit.dequeue()
                    visitedIndex.push(currentIndex)
                    findPathBFS(currentIndex, dgv, pathTracker, visitedIndex)

                if treasureCount is 0:
                    for c in tracker[arrToKey(currentIndex)]:
                        pathTracker.add(c)

                if TSP is true:
                    while queueToVisit is not empty:
                        currentIndex = queueToVisit.dequeue()
                        visitedIndex.push(currentIndex)
                        findPathBFS(currentIndex, dgv, pathTracker, visitedIndex)

                    reversedList = tracker[arrToKey(currentIndex)]
                    reversedList.reverse()
                    for c in reversedList:
                        d = opposite(c)
                        pathTracker.add(d)
```

4.1.2 findPathBFS

```
procedure findPathBFS(currentIndex, dgv, pathTracker, visitedIndex)
{I.S : melakukan pencarian berdasarkan currentIndex
F.S : semua tetangga node yang bersesuaian dicatat ke Queue untuk kemudian
diperiksa secara melebar}

checkPosition(currentIndex)
currentKey = arrToKey(currentIndex)
if isTop then
    if isLeft then
        checkRight(currentIndex, dgv, tracker[currentKey])
        checkDown(currentIndex, dgv, tracker[currentKey])
    else if isRight then
        checkDown(currentIndex, dgv, tracker[currentKey])
        checkLeft(currentIndex, dgv, tracker[currentKey])
    else
        checkRight(currentIndex, dgv, tracker[currentKey])
        checkDown(currentIndex, dgv, tracker[currentKey])
        checkLeft(currentIndex, dgv, tracker[currentKey])

else if isBottom then
    if isLeft then
        checkRight(currentIndex, dgv, tracker[currentKey])
        checkUp(currentIndex, dgv, tracker[currentKey])
    else if isRight then
        checkLeft(currentIndex, dgv, tracker[currentKey])
        checkUp(currentIndex, dgv, tracker[currentKey])
    else
        checkRight(currentIndex, dgv, tracker[currentKey])
        checkLeft(currentIndex, dgv, tracker[currentKey])
        checkUp(currentIndex, dgv, tracker[currentKey])

else
    if isLeft then
        checkRight(currentIndex, dgv, tracker[currentKey])
        checkDown(currentIndex, dgv, tracker[currentKey])
        checkUp(currentIndex, dgv, tracker[currentKey])
    else if isRight then
        checkDown(currentIndex, dgv, tracker[currentKey])
        checkLeft(currentIndex, dgv, tracker[currentKey])
        checkUp(currentIndex, dgv, tracker[currentKey])
    else
        checkRight(currentIndex, dgv, tracker[currentKey])
        checkDown(currentIndex, dgv, tracker[currentKey])
        checkLeft(currentIndex, dgv, tracker[currentKey])
        checkUp(currentIndex, dgv, tracker[currentKey])
```

4.1.3 DFSSolver

```
procedure DFSSolver(dgv, pathTracker)
{I.S : menerima input datagrid dan tracker untuk path
F.S : datagrid diwarnai sesuai aturan DFS dan pathTracker mencatat path yang dilalui}

enqueue(startIndex)
setCellColor(startIndex)
incrementAdjacent(startIndex)
currentIndex = startIndex

while queueToVisit is not empty and treasureCount > 0
    currentIndex = dequeue(queueToVisit)
    pushToStack(visitedIndex, currentIndex)
    findPathDFS(currentIndex, dgv, pathTracker, visitedIndex)

while queueToVisit is empty and treasureCount > 0
    currentIndex = popFromStack(visitedIndex)
    setCellColor(visitedIndex.peek())
    backtrackDFSHandler(currentIndex, visitedIndex)
    currentIndex = visitedIndex.peek()
    findPathDFS(currentIndex, dgv, pathTracker, visitedIndex)

if treasureCount == 0 and not TSP
    add trackPath to pathTracker

if TSP
    while queueToVisit is not empty and currentIndex != startIndex
        currentIndex = dequeue(queueToVisit)
        pushToStack(visitedIndex, currentIndex)
        findPathDFS(currentIndex, dgv, pathTracker, visitedIndex)

    while queueToVisit is empty and currentIndex != startIndex
        currentIndex = popFromStack(visitedIndex)
        setCellColor(visitedIndex.peek())
        backtrackDFSHandler(currentIndex, visitedIndex)
        currentIndex = visitedIndex.peek()
        findPathDFS(currentIndex, dgv, pathTracker, visitedIndex)

remove first half of trackPath from pathTracker
add trackPath to pathTracker
```

4.1.4 findPathDFS

```
procedure findPathDFS(currentIndex, dgv, pathTracker, visitedIndex)
{I.S : melakukan pencarian berdasarkan currentIndex
F.S : semua tetangga node yang bersesuaian dicatat ke Stack untuk kemudian
diperiksa secara mendalam}

    checkPosition(currentIndex)

    if isTop:
        if isLeft:
            checkTopLeft(currentIndex, dgv)
        else if isRight:
            checkTopRight(currentIndex, dgv)
        else:
            checkTopMiddle(currentIndex, dgv)

    else if isBottom:
        if isLeft:
            checkBottomLeft(currentIndex, dgv)
        else if isRight:
            checkBottomRight(currentIndex, dgv)
        else:
            checkBottomMiddle(currentIndex, dgv)

    else:
        if isLeft:
            checkMiddleLeft(currentIndex, dgv)
        else if isRight:
            checkMiddleRight(currentIndex, dgv)
        else:
            checkMiddle(currentIndex, dgv)
```

4.1.5 backtrackDFSHandler

```
procedure backtrackDFSHandler(currentIndex, visitedIndex
{I.S. Path DFS mengalami buntu karena tetangganya sudah dikunjungi semua
F.S. path DFS akan mundur ke node sebelumnya hingga menemukan path yang belum
dikunjungi}

if currentIndex[1] > visitedIndex.Peek()[1]:
    trackPath.Add("Left")
    nodeVisited++
else if currentIndex[1] < visitedIndex.Peek()[1]:
    trackPath.Add("Right")
    nodeVisited++
else if currentIndex[0] > visitedIndex.Peek()[0]:
    trackPath.Add("Up")
    nodeVisited++
else if currentIndex[0] < visitedIndex.Peek()[0]:
    trackPath.Add("Down")
    nodeVisited++
```

4.2 Struktur Data

Berikut merupakan struktur data beserta spesifikasi program yang digunakan dalam pembuatan program:

1. Queue

Implementasi struktur data Queue menggunakan collection pada kelas System.Collections.Generic. Queue digunakan untuk menyimpan antrian Node yang akan dikunjungi oleh program saat melakukan traversal dengan metode BFS. Berikut merupakan implementasi dari method yang digunakan.

- a. Enqueue, berfungsi untuk memasukkan Node ke dalam antrian dengan aturan FIFO atau *first in first out*.
- b. Dequeue, berfungsi untuk mengeluarkan Node dari antrian dengan aturan FIFO atau *first in first out*.

2. Stack

Implementasi struktur data Stack menggunakan collection pada kelas System.Collections.Generic. Stack digunakan untuk menyimpan tumpukan Node yang akan dikunjungi oleh program saat melakukan traversal dengan metode DFS. Berikut merupakan implementasi dari method yang digunakan.

- a. Push, berfungsi untuk memasukkan Node ke dalam tumpukan dengan aturan LIFO atau *last in first out*

- b. Pop, berfungsi untuk mengeluarkan Node dari tumpukan dengan aturan LIFO atau *last in first out*
 - c. Peek, berfungsi untuk mengakses elemen tumpukan teratas tanpa mengeluarkan elemen teratas tersebut
3. List
- Struktur data list merupakan struktur data dinamis yang disediakan oleh .NET Framework. Beberapa method yang digunakan diantaranya adalah sebagai berikut:
- a. List<T>.Add(T data)
Digunakan untuk menambahkan data pada list
 - b. List<T>.Clear()
Menghapus semua elemen pada list
 - c. List<T>.Count
Mengembalikan nilai jumlah elemen dalam list
 - d. List<T>.Reverse()
Membalik urutan-urutan elemen pada list

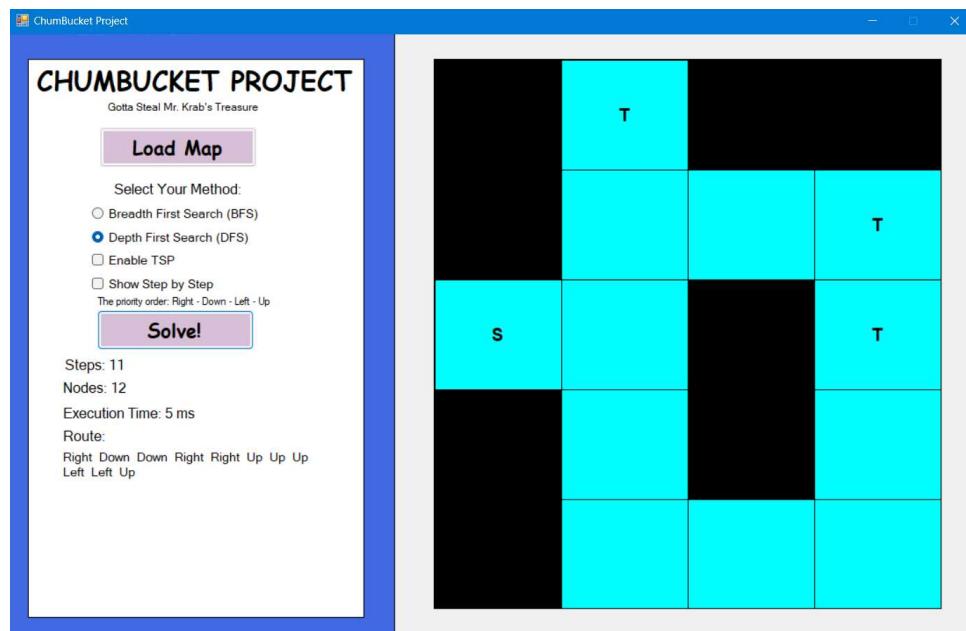
4.3 Tata Cara Penggunaan Program

Pengguna dapat menjalankan program dengan mengakses folder bin dan menjalankan file “ChumBucketProject.exe”. Setelah menjalankan program, pengguna akan dihadapkan dengan beberapa tombol salah satunya untuk melakukan input file pada button Load. Setelah itu, pengguna juga dapat memilih algoritma yang ingin digunakan dalam menjalankan program. Setelah melakukan input, dapat menekan tombol Solve untuk mulai melakukan pencarian. Pada saat pencarian, grid yang tersedia akan berubah warna dan ada panduan proses perjalanan *path*. Berikut merupakan tampilan dari program atau desktop application yang telah kami buat.



Gambar 4.1 Tampilan Awal Program

(sumber: Dokumen Pribadi)



Gambar 4.2 Tampilan Akhir Program setelah Melakukan Pencarian dengan DFS

(sumber: Dokumen Pribadi)

4.4 Hasil Pengujian

4.4.1 Pengujian 1

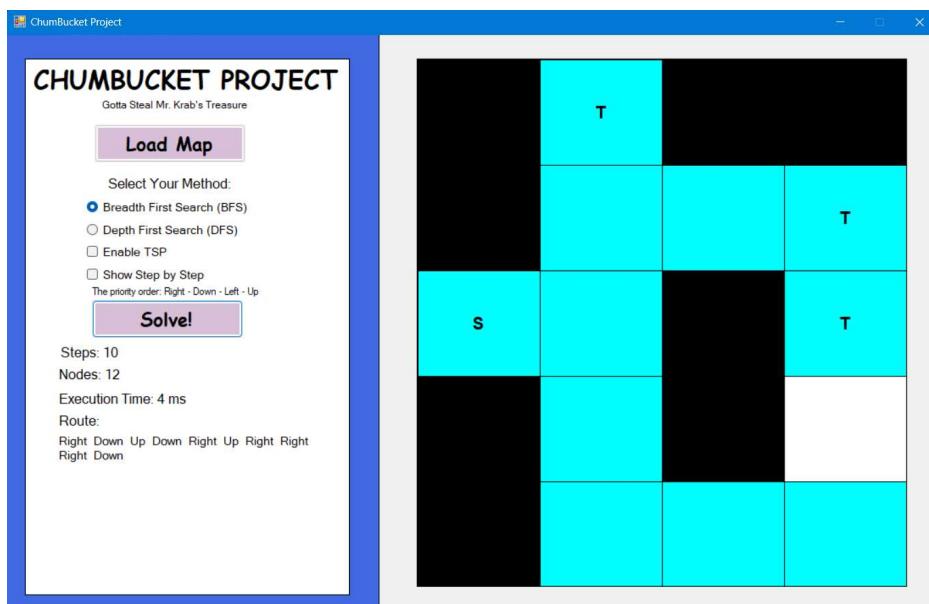
Berikut merupakan kasus untuk pengujian 1 dengan file sampel-1.txt

sample-1	
File	Edit
X T X X	
X R R T	
K R X T	
X R X R	
X R R R	

Gambar 4.3 Kasus Pengujian 1

(Sumber: Dokumen Pribadi)

Penelusuran menggunakan BFS

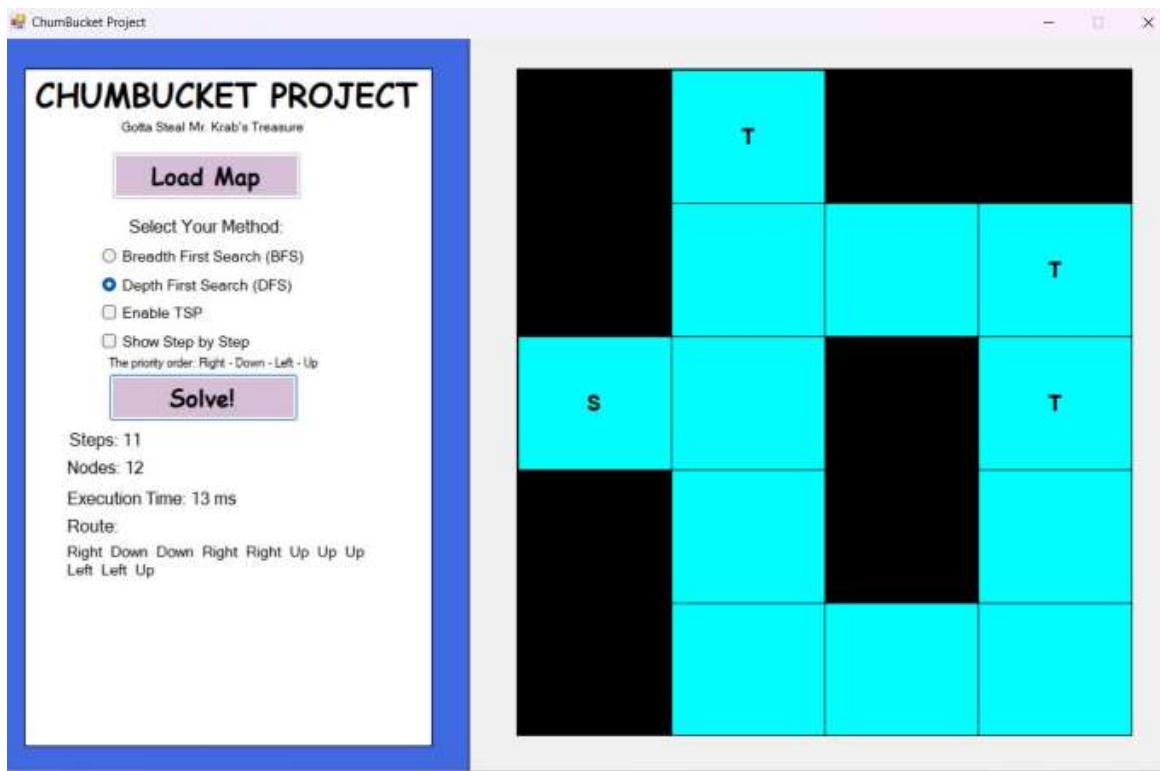


Gambar 4.4 Penelusuran Menggunakan BFS Kasus Pengujian 1

(Sumber: Dokumen Pribadi)

Dengan BFS, rute yang dicatat adalah proses berjalanannya grid sesuai dengan algoritma pencarian melebar, sehingga kurang terlihat terurut jika dilihat secara langsung. Paling baik adalah dengan menceklis show step by step agar terlihat proses langkah-langkahnya.

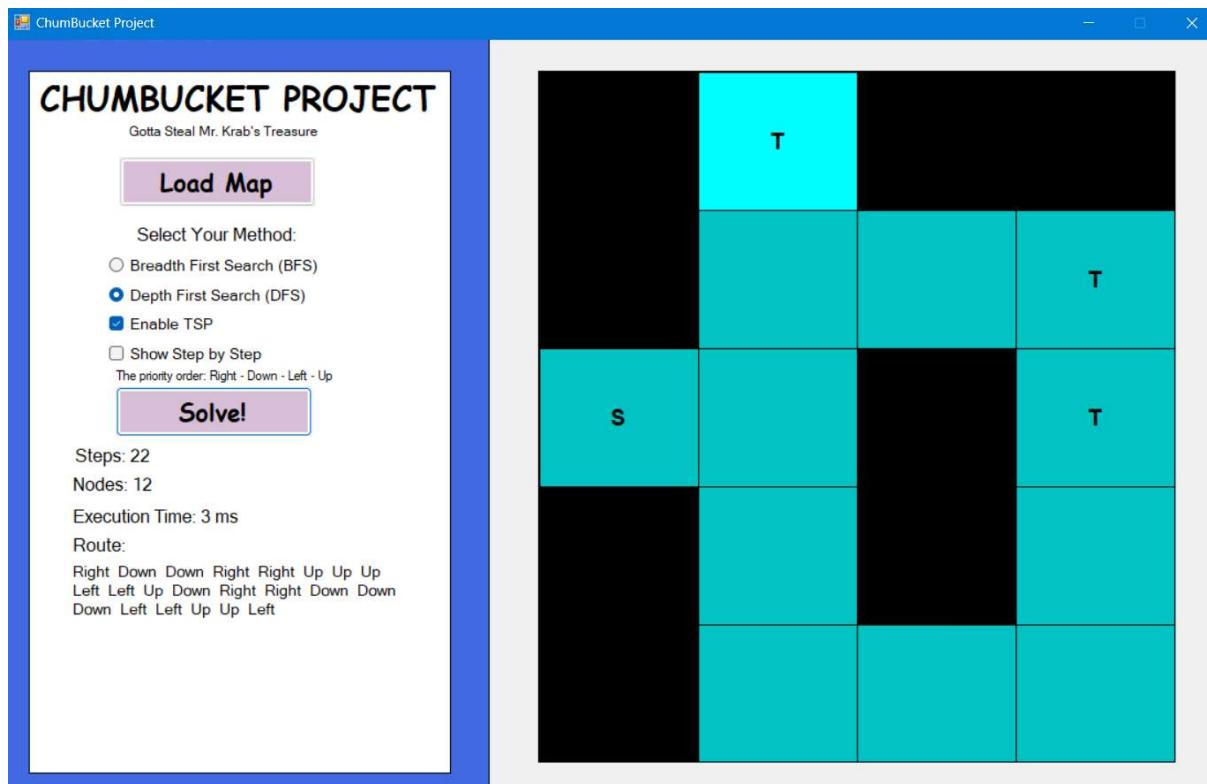
Penelusuran menggunakan DFS



Gambar 4.5 Penelusuran Menggunakan DFS Kasus Pengujian 1

(Sumber: Dokumen Pribadi)

Penelusuran dengan TSP



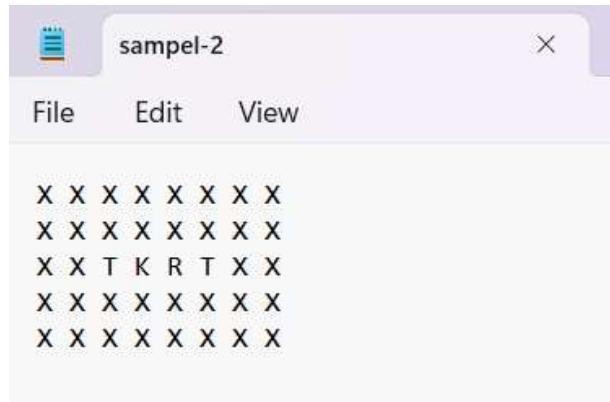
Gambar 4.5 Penelusuran Menggunakan DFS Kasus Pengujian 1

(Sumber: Dokumen Pribadi)

Terlihat warna yang lebih gelap, artinya dikunjungi lebih dari satu kali (dalam hal ini 2 kali) untuk kembali ke titik Start. Algoritma TSP ini memang belum menghasilkan solusi yang optimal.

4.4.2 Pengujian 2

Berikut merupakan kasus untuk pengujian 2 dengan file sampel-2.txt



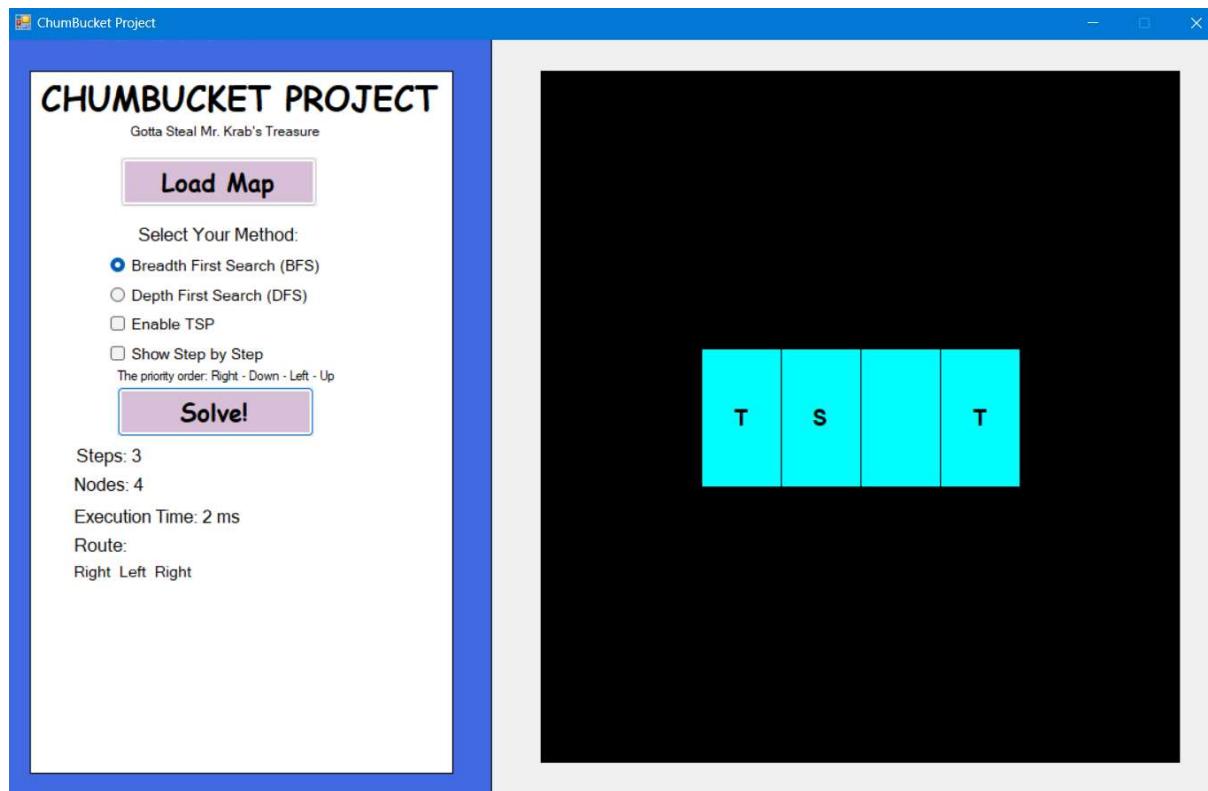
The screenshot shows a text editor window with the title bar 'sampel-2'. The menu bar includes 'File', 'Edit', and 'View'. The main content area displays a 5x8 grid of characters, where each row starts with an 'X' and ends with an 'X'. The second row contains two 'T's. The third row contains two 'R's. The fourth row contains two 'X's. The fifth row contains two 'X's. The grid is as follows:

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

Gambar 4.7 Kasus Pengujian 2

(Sumber: Dokumen Pribadi)

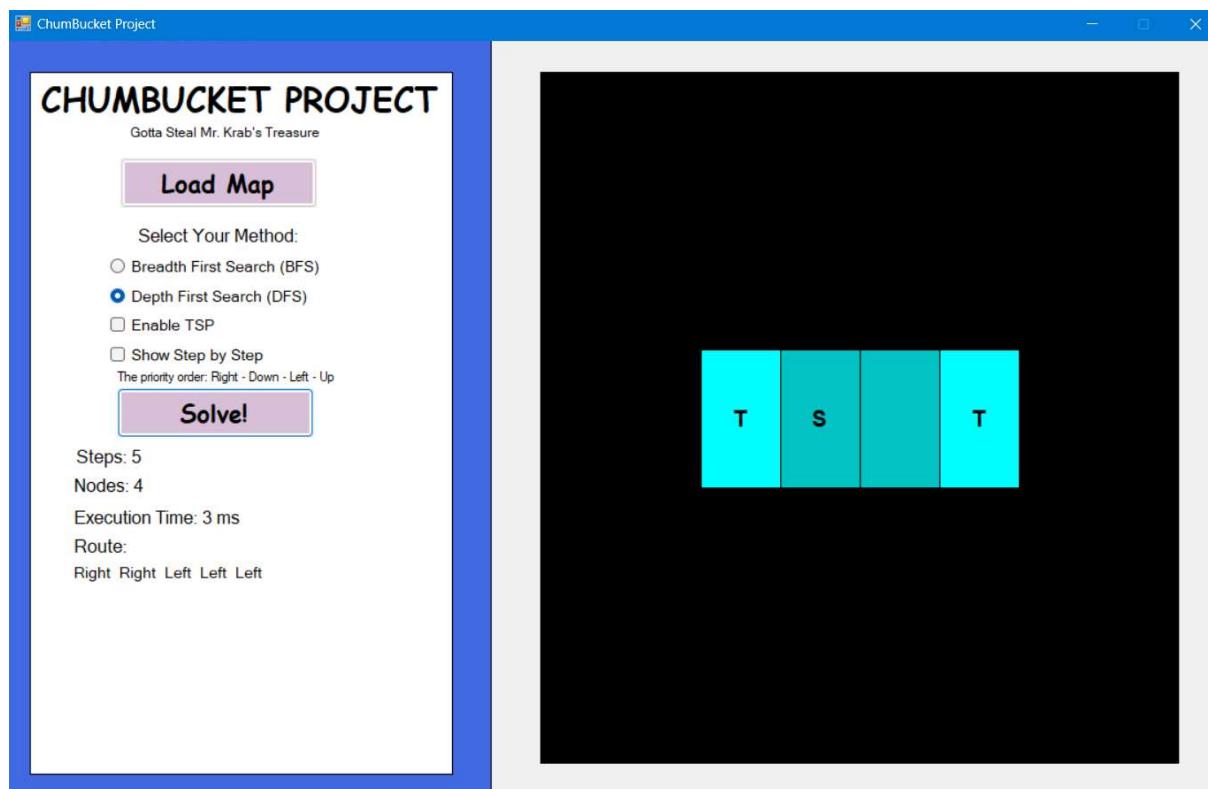
Penelusuran menggunakan BFS



Gambar 4.8 Penelusuran Menggunakan BFS Kasus Pengujian 2

(Sumber: Dokumen Pribadi)

Penelusuran menggunakan DFS

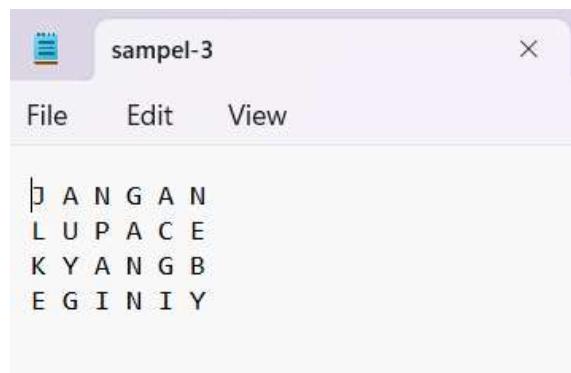


Gambar 4.9 Penelusuran Menggunakan DFS Kasus Pengujian 2

(Sumber: Dokumen Pribadi)

4.4.3 Pengujian 3

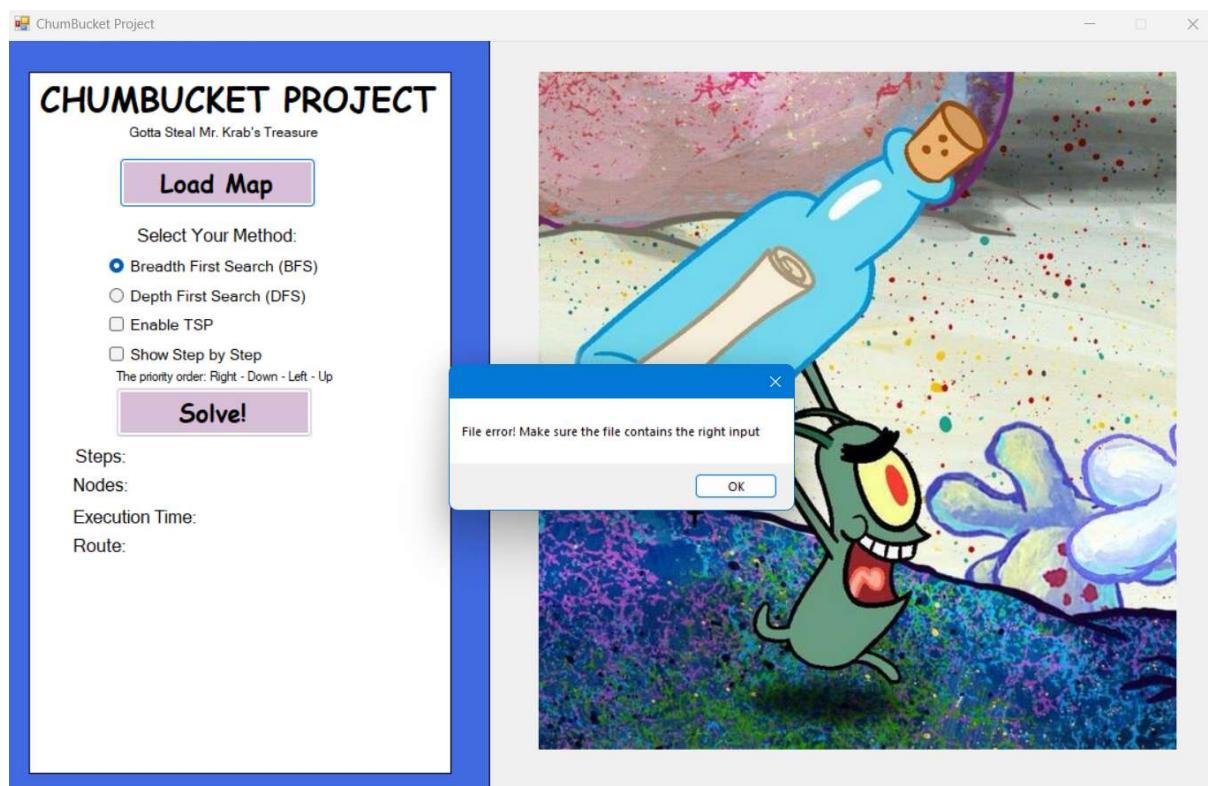
Berikut merupakan kasus untuk pengujian 3 dengan file sampel-3.txt yang tidak sesuai dengan ketentuan file testcase.



Gambar 4.10 Kasus Pengujian 3

(Sumber: Dokumen Pribadi)

Penelusuran menggunakan BFS ataupun DFS akan menghasilkan pesan error yang sama

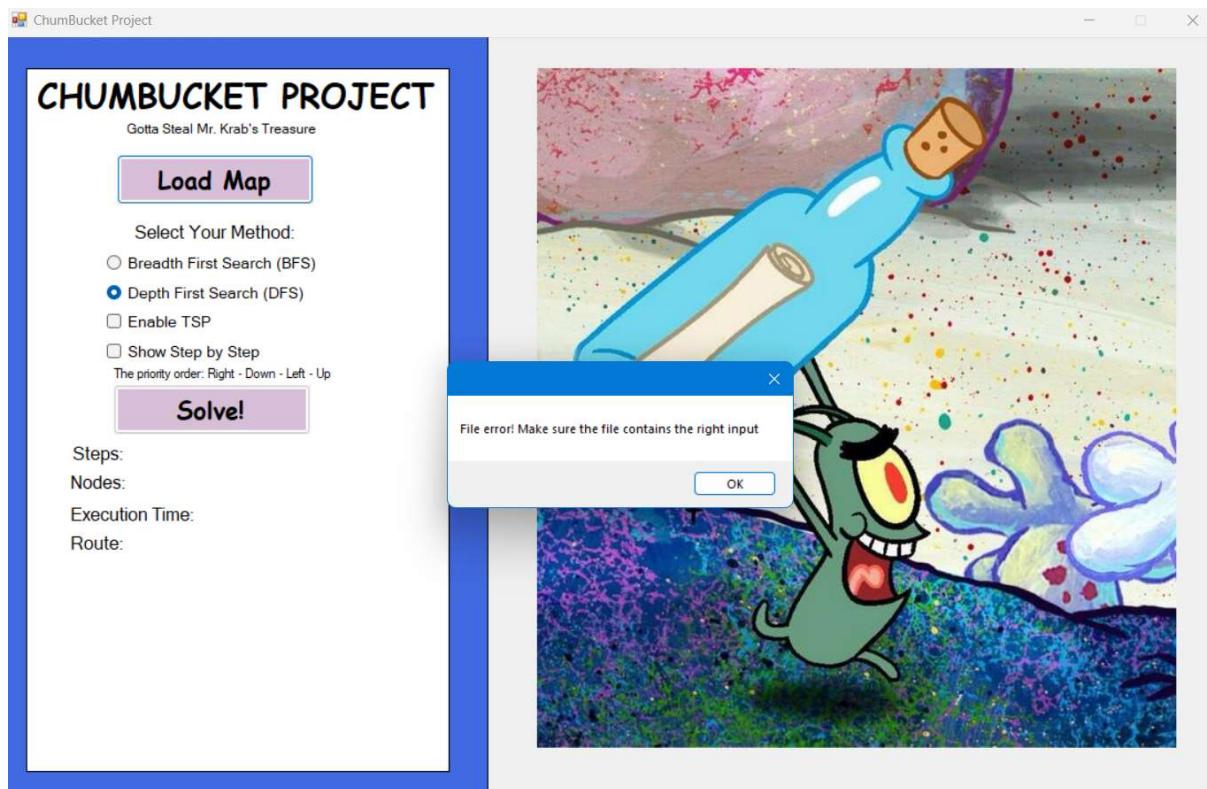


Gambar 4.11 Penelusuran Menggunakan BFS Kasus Pengujian 3

(Sumber: Dokumen Pribadi)

Jika file mengandung elemen yang tidak tepat, akan muncul validasi berupa pesan error dan pengguna dapat memasukkan kembali file lain yang tepat.

Penelusuran menggunakan DFS



Gambar 4.12 Penelusuran Menggunakan DFS Kasus Pengujian 3

(Sumber: Dokumen Pribadi)

4.4.4 Pengujian 4

Berikut merupakan kasus untuk pengujian 4 dengan file sampel-4.txt

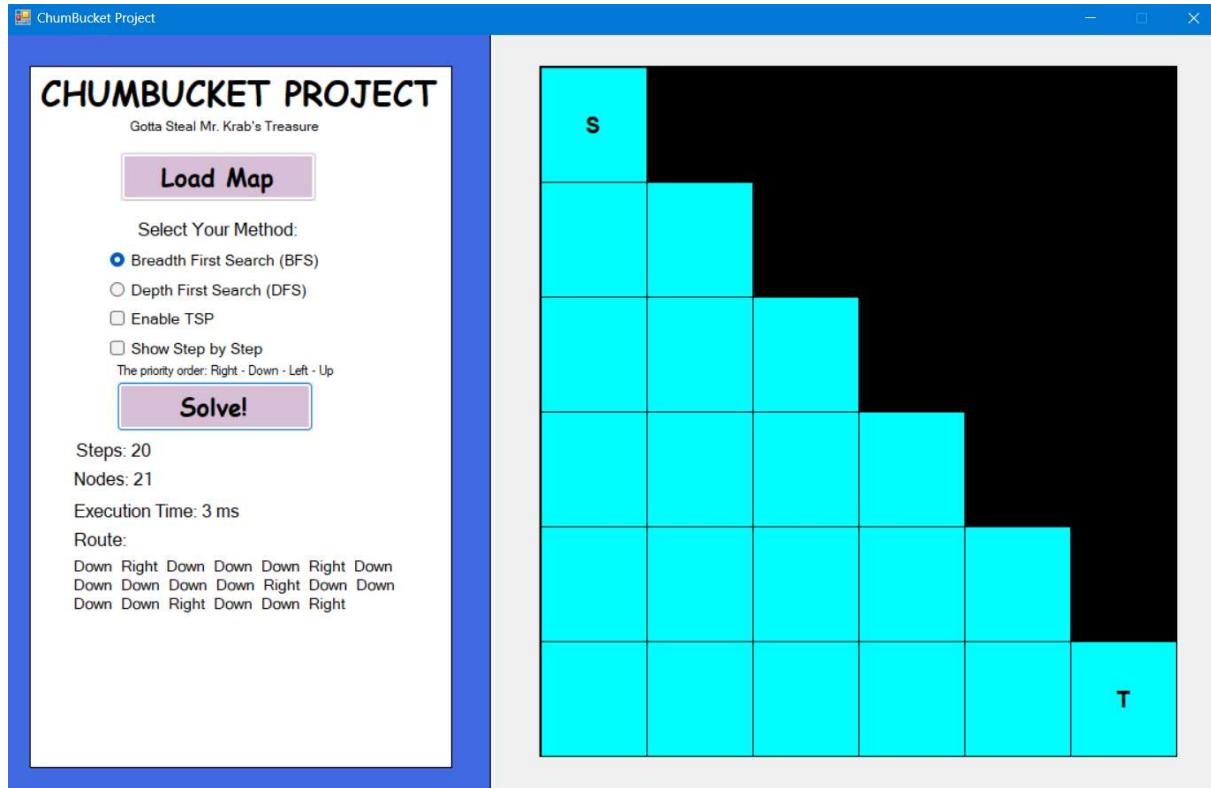
A screenshot of a Windows-style text editor window titled 'sampel-4'. The menu bar includes 'File', 'Edit', and 'View'. The main text area contains the following grid of characters:

```
K X 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
```

Gambar 4.13 Kasus Pengujian 4

(Sumber: Dokumen Pribadi)

Penelusuran menggunakan BFS

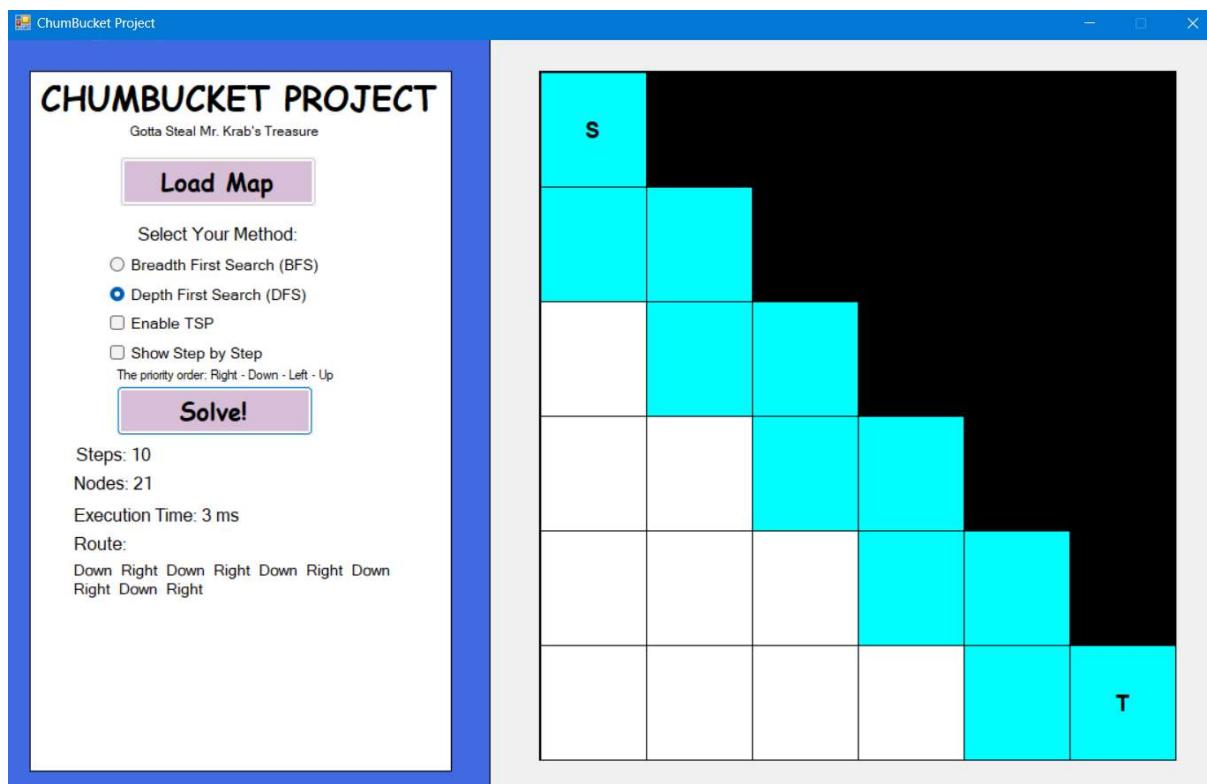


Gambar 4.14 Penelusuran Menggunakan BFS Kasus Pengujian 4

(Sumber: Dokumen Pribadi)

Dengan BFS, semua node akan diperiksa karena berdasarkan pencarian secara melebar.

Penelusuran menggunakan DFS



Gambar 4.15 Penelusuran Menggunakan DFS Kasus Pengujian 4

(Sumber: Dokumen Pribadi)

Dengan DFS, pencarian pada kasus seperti ini akan lebih efektif dan lebih cepat.

4.4.5 Pengujian 5

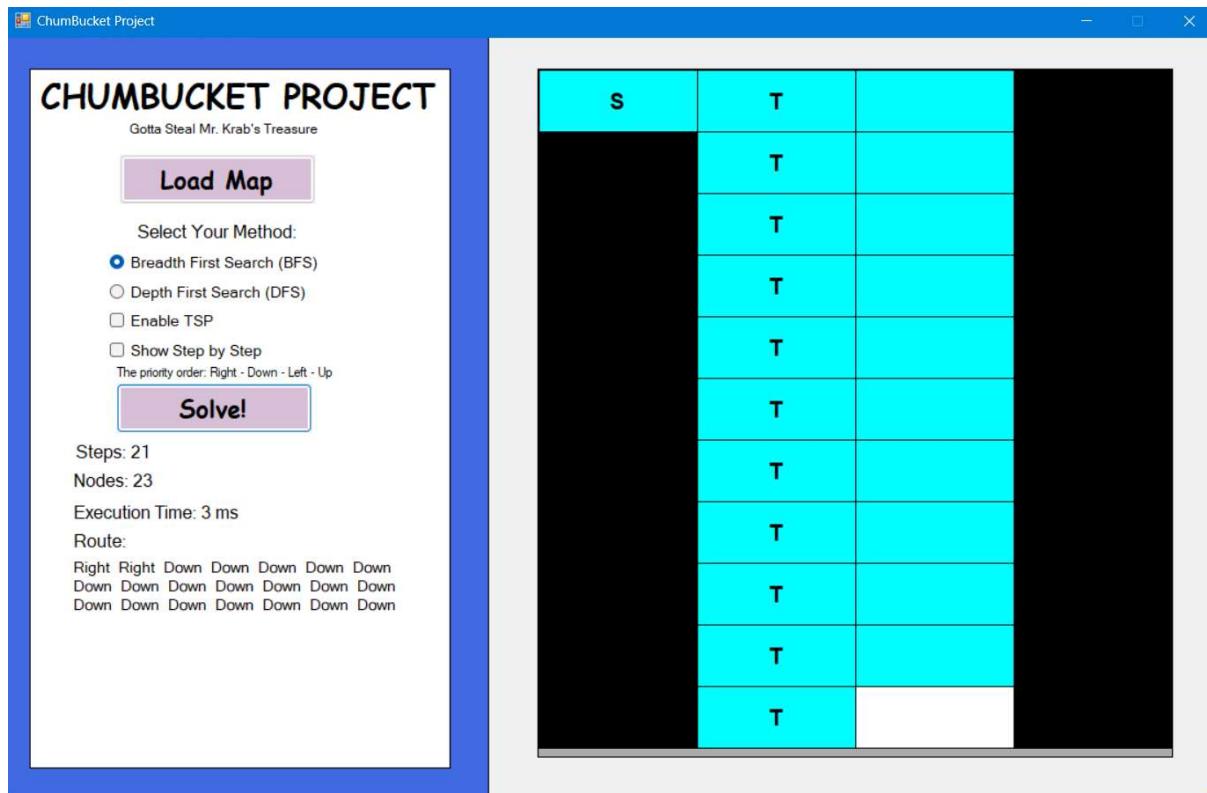
Berikut merupakan kasus untuk pengujian 5 dengan file sampel-5.txt

A screenshot of a Microsoft Word document window titled "sampel-5". The menu bar includes "File", "Edit", and "View". The document content consists of a sequence of 15 lines, each containing a string of characters: K T R X, X T R X, and X T R X.

Gambar 4.16 Kasus Pengujian 5

(Sumber: Dokumen Pribadi)

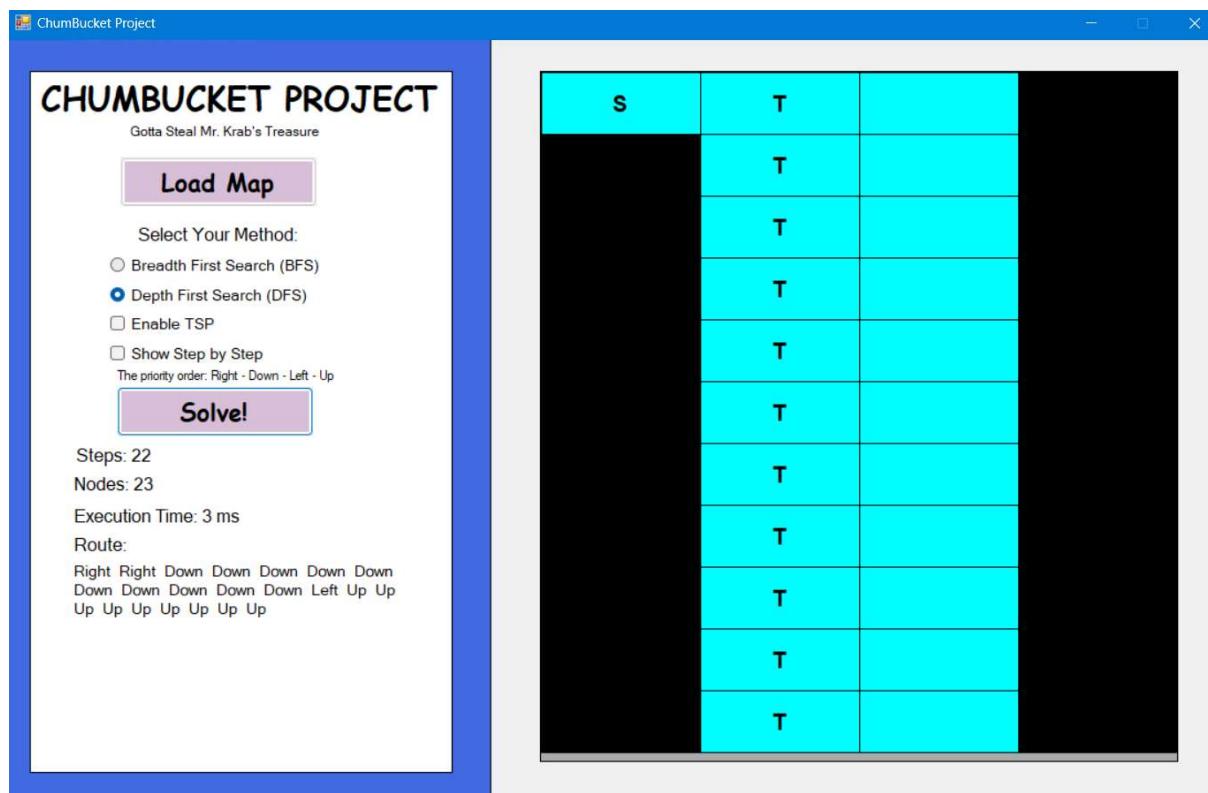
Penelusuran menggunakan BFS



Gambar 4.17 Penelusuran Menggunakan BFS Kasus Pengujian 5

(Sumber: Dokumen Pribadi)

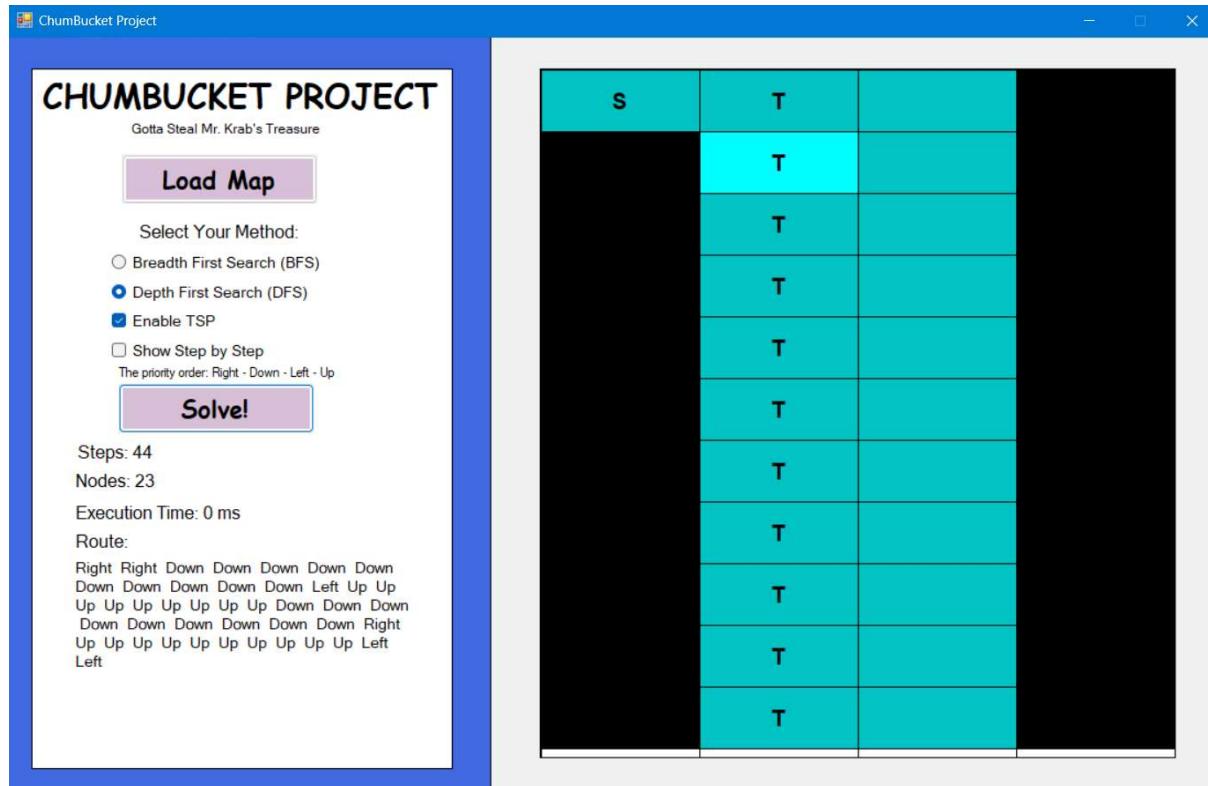
Penelusuran menggunakan DFS



Gambar 4.18 Penelusuran Menggunakan DFS Kasus Pengujian 5

(Sumber: Dokumen Pribadi)

Penelusuran dengan TSP



Gambar 4.19 Penelusuran Menggunakan DFS dan TSP Kasus Pengujian 5

(Sumber: Dokumen Pribadi)

4.5 Analisis Desain Solusi Algoritma BFS dan DFS berdasarkan Hasil Pengujian

Berdasarkan hasil pengujian, algoritma BFS dan DFS mengalami perbedaan baik antara proses pencarinya, path yang dikunjungi, waktu penelusuran yang dibutuhkan, dan bahkan jumlah langkahnya. Pada pengujian pertama misalnya. Algoritma BFS akan lebih menguntungkan dibandingkan dengan DFS, dilihat dari waktu algoritma dan jumlah langkah yang diperlukan. Pada pengujian kedua juga terlihat perbedaan mengenai jumlah langkah dan waktu algoritmanya.

Kesimpulannya, ini semua didasarkan pada map yang diberikan dan dengan prioritas yang ditentukan. Tidak ada cara yang pasti apakah BFS atau DFS memberikan hasil terbaik.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang didapatkan dari hasil pengaplikasian algoritma BFS dan DFS dalam implementasi *Maze Treasure Hunt* adalah sebagai berikut:

1. Aplikasi *Maze Treasure Hunt* yang dihasilkan mampu mengimplementasikan algoritma BFS untuk melakukan pencarian rute dengan baik.
2. Aplikasi *Maze Treasure Hunt* yang dihasilkan mampu mengimplementasikan algoritma DFS untuk melakukan pencarian rute dengan baik.
3. Aplikasi yang telah dibuat mampu menampilkan waktu eksekusi algoritma.
4. Aplikasi yang telah dibuat mampu menampilkan jumlah langkah yang dilewati saat pencarian rute.
5. Aplikasi yang telah dibuat mampu menampilkan jumlah node yang ada.
6. Aplikasi yang telah dibuat mampu menampilkan jalur rute mana saja yang telah dilewati.
7. Aplikasi yang telah dibuat mampu mengimplementasikan toggle untuk persoalan TSP.

5.2 Saran

Saran yang dapat kami berikan dari pengembangan aplikasi ini adalah penulisan pseudocode tampak kurang perlu dikarenakan program yang lumayan panjang dan membaca program lebih mudah daripada membaca pseudocode dengan asumsi program sudah *well commented*. Selain itu, pengembangan *desktop application* dapat dikembangkan menggunakan pustaka yang berbeda sehingga dapat membuat desktop application yang lebih *versatile*. Saran lainnya adalah akan lebih baik untuk mengerjakan penulisan struktur data dengan algoritma secara bersamaan karena besar kemungkinan ada kesalahpahaman antara dua orang yang mengerjakan hal tersebut.

5.3 Refleksi

Setelah menyelesaikan tugas besar kedua IF2211 Strategi Algoritma, kami dapat merefleksikan bahwa komunikasi antar anggota kelompok berjalan cukup baik sehingga tidak terjadi miskomunikasi dan kesalahpahaman dalam penggerjaan tugas besar ini, sebelum dimulainya penggerjaan tugas besar ini, sudah dilakukan diskusi untuk membahas pembagian kerja untuk setiap anggota kelompok, dan terakhir kami menyadari perlunya mempelajari

algoritma *Breadth-First Search (BFS)* dan *Depth-First Search (DFS)* dalam menyelesaikan suatu permasalahan.

5.4 Komentar

Dari tugas besar IF2211 Strategi Algoritma yang telah diberikan, sudah berjalan dengan lancar dan baik untuk koordinasi antar anggota kelompok. Program yang dibuat juga berhasil melakukan penelusuran maze dengan algoritma *breadth-first search (BFS)* dan *depth-first search (DFS)* beserta dengan jumlah langkah, jumlah node, serta waktu eksekusi program. Untuk spesifikasi tugas yang diberikan sudah cukup jelas.

DAFTAR PUSTAKA

<https://docs.microsoft.com/en-us/dotnet/csharp/> diakses pada 15 Februari 2023

<https://learn.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?toc=%2Fvisualstudio%2Fget-started%2Fcsharp%2Ftoc.json&bc=%2Fvisualstudio%2Fget-started%2Fcsharp%2Fbreadcrumb%2Ftoc.json&view=vs-2022> diakses pada 15 Februari 2023

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>
diakses pada 15 Februari 2023

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>
diakses pada 15 Februari 2023

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf> diakses pada 17 Februari 2023

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian2.pdf> diakses pada 17 Februari 2023

LAMPIRAN

Link Github:

https://github.com/blixa-rd/Tubes2_ChumBucket.git