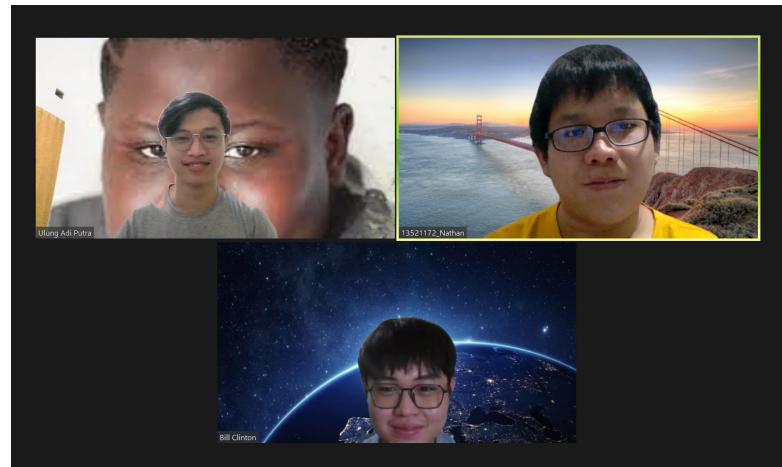


# **LAPORAN TUGAS BESAR 2**

## **IF2211 STRATEGI ALGORITMA**

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



**oleh**

**Bill Clinton  
Ulung Adi Putra  
Nathan Tenka**

**13521064  
13521122  
13521172**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
BANDUNG  
2022/2023**

## DAFTAR ISI

<b>BAB I</b>	<b>1</b>
<b>DESKRIPSI TUGAS</b>	<b>1</b>
<b>BAB II</b>	<b>11</b>
<b>LANDASAN TEORI</b>	<b>11</b>
<b>2.1 Graph Traversal, BFS, DFS Secara Umum</b>	<b>11</b>
2.1.1 Graph Traversal	11
2.1.2 Breadth First Search (BFS)	11
2.1.3 Depth First Search (DFS)	13
<b>2.2 C# Desktop Application Development</b>	<b>14</b>
<b>BAB III</b>	<b>16</b>
<b>APLIKASI STRATEGI BFS/DFS</b>	<b>16</b>
<b>3.1 Langkah-Langkah Pemecahan Masalah</b>	<b>16</b>
<b>3.2 Pemetaan Persoalan Menjadi Elemen BFS dan DFS</b>	<b>17</b>
3.2.1 Breadth First Search (BFS)	17
3.2.2 Depth First Search (DFS)	17
<b>3.3 Contoh Ilustrasi Kasus Lain</b>	<b>17</b>
<b>BAB IV</b>	<b>20</b>
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>20</b>
<b>4.1 Implementasi Program</b>	<b>20</b>
4.1.1 Implementasi Algoritma BFS	20
4.1.2 Implementasi Algoritma DFS	23
<b>4.2 Penjelasan Struktur Data dan Spesifikasi Program</b>	<b>25</b>
<b>4.3 Tata Cara Penggunaan Program</b>	<b>28</b>
4.4 Hasil Pengujian dan Analisis	28
<b>BAB V</b>	<b>46</b>
<b>KESIMPULAN DAN SARAN</b>	<b>46</b>
<b>5.1 Kesimpulan</b>	<b>46</b>
<b>5.2 Saran</b>	<b>46</b>
5.3 Refleksi	46
5.4 Tanggapan	47
<b>DAFTAR PUSTAKA</b>	<b>48</b>
<b>LAMPIRAN</b>	<b>49</b>
<b>Link Repository Program</b>	<b>50</b>
<b>Link Video Demo Youtube</b>	<b>50</b>

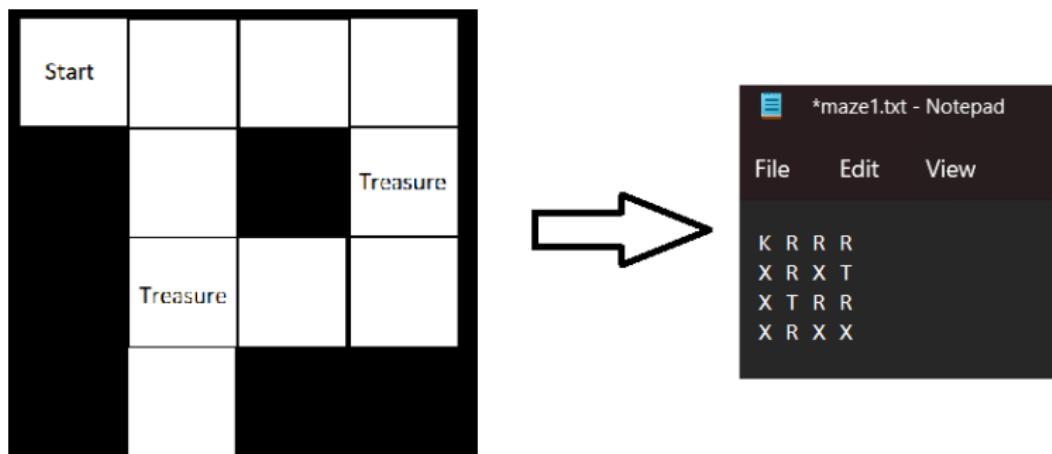
# BAB I

## DESKRIPSI TUGAS

Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi dengan GUI sederhana yang dapat mengimplementasikan BFS dan DFS untuk mendapatkan rute memperoleh seluruh treasure atau harta karun yang ada. Program dapat menerima dan membaca input sebuah *file txt* yang berisi *maze* yang akan ditemukan solusi rute mendapatkan *treasure*-nya. Untuk mempermudah, batasan dari *input maze* cukup berbentuk segiempat 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

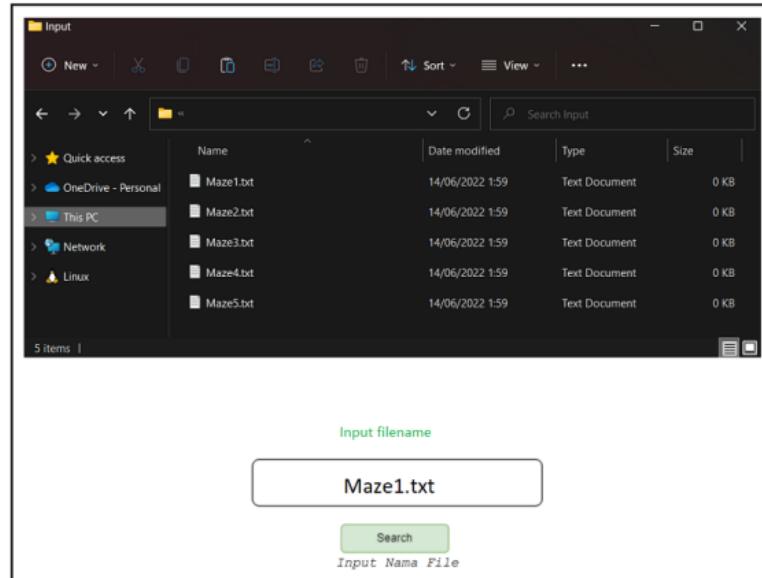
Contoh *file input* :



Gambar 2. 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.

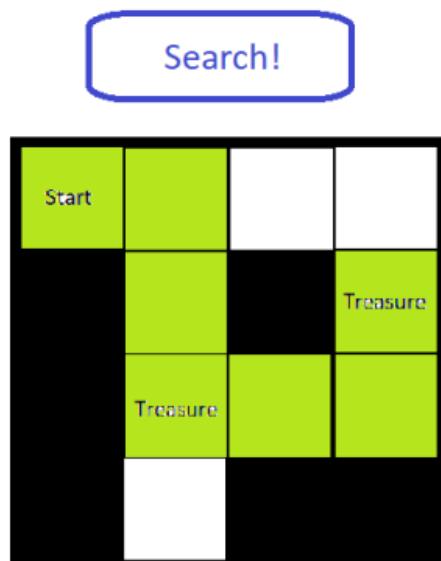
Contoh *input* aplikasi :



Gambar 3. 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 :

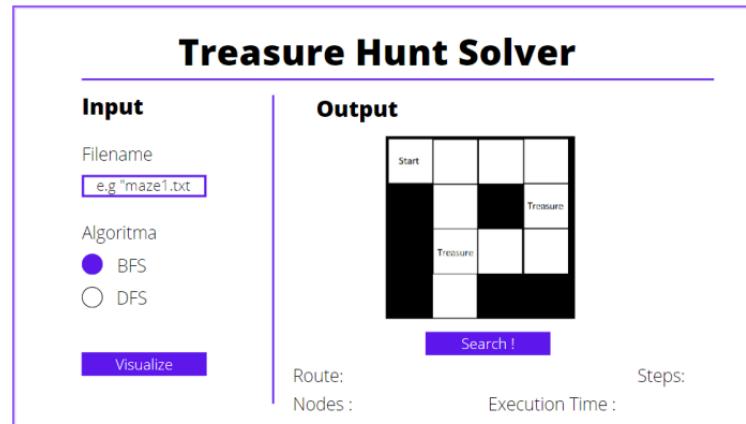


Gambar 4. Contoh output program untuk gambar 2

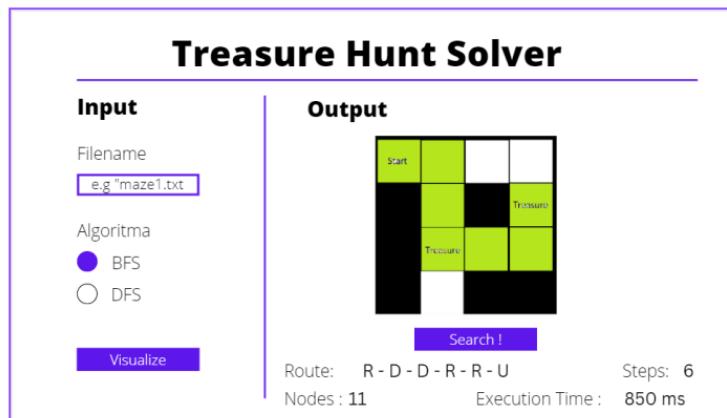
Setelah program melakukan pembacaan *input*, program akan memvisualisasikan *gridnya* terlebih dahulu tanpa pemberian rute solusi. Hal tersebut dilakukan agar pengguna dapat mengerjakan terlebih dahulu *treasure hunt* secara manual jika diinginkan. Kemudian, program menyediakan tombol *solve* untuk mengeksekusi algoritma BFS dan DFS. 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 5. Tampilan Program Sebelum dicari solusinya



Gambar 6. 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.

Lain – lain:

1. Anda dapat menambahkan fitur-fitur lain yang menunjang program yang anda buat (unsur kreativitas).

2. Tugas dikerjakan berkelompok, minimal 2 orang dan maksimal 3 orang, boleh lintas kelas namun tidak diperbolehkan sekelompok dengan orang yang sama dengan tubes ataupun tucil stima sebelumnya
3. Semua kelompok harap mengisi data kelompok mereka pada tautan berikut:  
<https://bit.ly/KelompokTubes2Stima>
4. Program dibuat dengan bahasa C# dengan kakas Visual Studio .NET, pelajarilah C# desktop development, **disarankan untuk menggunakan Visual Studio** untuk mempermudah penggerjaan
5. Program harus modular dan mengandung komentar yang jelas.
6. Beri nama aplikasi anda tersebut dengan nama-nama yang menarik dan mudah diingat.
7. Dilarang menggunakan kode program yang diunduh dari *Internet*. Mahasiswa harus membuat program sendiri, tetapi belajar dari program yang sudah ada tidak dilarang.
8. Batas akhir pengumpulan tugas adalah **Jumat, 24 Maret 2023 23:59 WIB**. Keterlambatan dalam mengumpulkan akan diberi penalti pengurangan skor yang cukup signifikan.
9. Semua pertanyaan menyangkut tugas ini dapat dikomunikasikan lewat QnA yang bisa diakses pada [bit.ly/TugasStimaQnA](https://bit.ly/TugasStimaQnA)
10. Contoh *test case* untuk tubes ini dapat diunduh melalui tautan berikut:  
[https://drive.google.com/drive/folders/1y-hrW6U2w5HoWdkf722yinZMOJVvLQoF?usp=share\\_link](https://drive.google.com/drive/folders/1y-hrW6U2w5HoWdkf722yinZMOJVvLQoF?usp=share_link)

11. **Bonus (nilai maksimal 5):** Setiap kelompok membuat video aplikasi yang mereka buat kemudian mengunggahnya ke Youtube. Video yang dibuat harus memiliki audio dan menampilkan wajah dari setiap anggota kelompok. Pada waktu demo aplikasi di depan asisten, mahasiswa mengakses video Youtube tersebut dan memutarnya di depan asisten. Beberapa contoh video tubes tahun-tahun sebelumnya dapat dilihat di YouTube dengan menggunakan kata kunci “Tubes Stima”, “Tugas besar stima”, “strategi algoritma”, dll.

12. Demo akan dilakukan, tunggu informasi lanjut setelah waktu peng交rajan tugas berakhir

13. Setiap anggota kelompok harus memahami seluruh program, termasuk bagian yang bukan bagian mereka

14. Program disimpan dalam folder **Tubes2\_NamaKelompok**. Berikut adalah struktur dari isi folder tersebut :

- a. Folder **src** berisi *source code*
- b. Folder **test** berisi *input file txt* untuk *maze*
- c. Folder **bin** berisi **executable** code / hasil *build* dari program C#
- d. Folder **doc** berisi **laporan tugas besar** dengan *format nama\_kelompok.pdf*
- e. **README** untuk tata cara penggunaan yang minimal berisi :
  - i. Deskripsi singkat program yang dibuat
  - ii. *Requirement* program dan instalasi *module/package* tertentu bila ada
  - iii. Langkah meng-*compile* program jika diperlukan

- iv. Cara menggunakan program
  - v. *Author / identitas pembuat*
- f. Catatan untuk **README agar dibuat selengkap-lengkapnya**. Anggap asisten sebagai orang awam yang tidak tahu apa-apa. Jangan sampai asisten mencari tahu sendiri bagaimana cara menjalankan program Anda.
15. Program disimpan pada *repository* Github yang *di-private* sebelum *deadline* dan *di-public* setelah *deadline* pengumpulan. Kelompok juga diminta untuk mengundang asisten sebagai *collaborator* dalam *repository*. Kelompok dapat melakukan submisi pengumpulan dengan cara mensubmit *file* laporan kalian pada tautan yang tersedia pada sheet QnA.

## **BAB II**

### **LANDASAN TEORI**

#### **2.1 *Graph Traversal, BFS, DFS Secara Umum***

##### **2.1.1 *Graph Traversal***

*Graph Traversal* adalah persoalan yang bertujuan mengunjungi simpul dalam suatu graf dengan cara yang sistematis. Graf bisa dianggap sebagai representasi dari persoalan, dan traversal graf bisa dianggap sebagai proses pencarian solusi.

Algoritma pencarian solusi dengan traversal graf ada dua jenis. Pertama, ada yang tanpa informasi (*uninformed/blind search*). Sesuai namanya, algoritma ini menelusuri graf tanpa memiliki informasi tambahan yang dapat menunjukkan apakah suatu jalur mendekati solusi atau tidak. Contohnya adalah DFS,BFS, *Depth Limited Search*, dll. Kedua, ada yang dengan informasi (*informed search*). Algoritma ini menelusuri graf dengan memiliki informasi yang menunjukkan apakah suatu jalur mendekati solusi atau tidak.

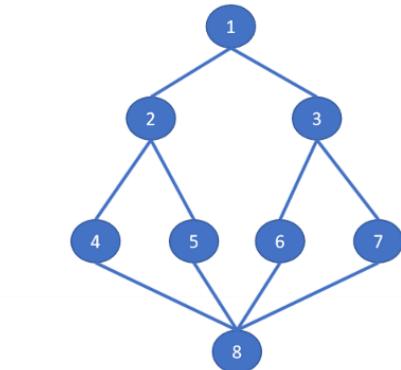
Ada 2 jenis pendekatan saat pencarian solusi. Ada yang menggunakan graf statis, yaitu graf sudah terbentuk sebelum pencarian dilakukan. Untuk kasus ini, biasanya persoalannya memang dalam bentuk graf. Jenis graf lain adalah graf dinamis, yaitu graf terbentuk saat proses pencarian dilakukan. Contohnya saat pencarian solusi dibentuk pohon pencarian.

##### **2.1.2 Breadth First Search (BFS)**

BFS atau *Breadth First Search* adalah algoritma untuk melakukan pencarian jalur atau lintasan pada suatu graf. Pada algoritma ini, akan diprioritaskan untuk menelusuri simpul pada *level* yang sama dengan simpul awal terlebih dahulu

sebelum menelusuri pada simpul dengan level berikutnya. Berikut adalah contoh tahap penelusuran dan *pseudocode* umumnya :

## BFS: Ilustrasi



Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

Urutan simpul yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

```

procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.

  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke layar
}

Deklarasi
w : integer
q : antrian;

procedure BuatAntrian(input/output q : antrian)
{ membuat antrian kosong, kepala(q) diisi 0 }

procedure MasukAntrian(input/output q:antrian, input v:integer)
{ memasukkan v ke dalam antrian q pada posisi belakang }

procedure HapusAntrian(input/output q:antrian,output v:integer)
{ menghapus v dari kepala antrian q }

function AntrianKosong(input q:antrian) → boolean
{ true jika antrian q kosong, false jika sebaliknya }

Algoritma:
BuatAntrian(q)           { buat antrian kosong }
write(v)                 { cetak simpul awal yang dikunjungi }
dikunjungi[v]←true     { simpul v telah dikunjungi, tandai dengan true}
MasukAntrian(q,v)        { masukkan simpul awal kunjungan ke dalam antrian }

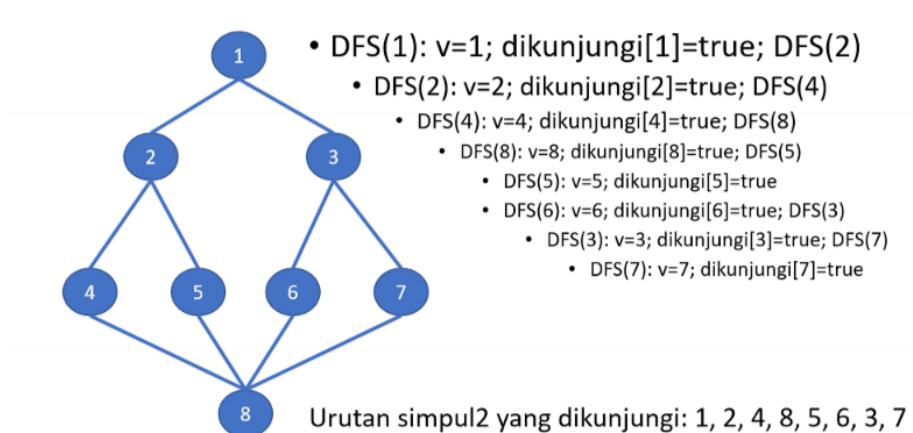
{ kunjungi semua simpul graf selama antrian belum kosong }
while not AntrianKosong(q) do
  HapusAntrian(q,v)      { simpul v telah dikunjungi, hapus dari antrian }
  for tiap simpul w yang bertetangga dengan simpul v do
    if not dikunjungi[w] then
      write(w)             { cetak simpul yang dikunjungi }
      MasukAntrian(q,w)
      dikunjungi[w]←true
    endif
  endfor
endwhile
{ AntrianKosong(q) }
  
```

### 2.1.3 Depth First Search (DFS)

DFS atau *Depth First Search* adalah algoritma pencarian jalur seperti BFS.

Namun, DFS mengunjungi mengunjungi simpul tetangga di *level* yang lebih dalam dulu dibanding mengunjungi semua simpul di *level* yang sama seperti BFS. Jika sudah tidak ada tetangga yang bisa dikunjungi, algoritma akan melakukan *backtrack* ke simpul sebelumnya dan mencari tetangga lain dari simpul tersebut.

Contoh alur penelusuran dan *pseudocode* umumnya adalah seperti berikut :



```
procedure DFS(input v:integer)
(Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS)

Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi ditulis ke layar
)
Deklarasi
w : integer

Algoritma:
  write(v)
  dikunjungi[v]←true
  for w←1 to n do
    if A[v,w]=1 then (simpul v dan simpul w bertetangga )
      if not dikunjungi[w] then
        DFS(w)
      endif
    endif
  endfor
```

↗ rekursif

## 2.2 C# Desktop Application Development

C# merupakan bahasa pemrograman berbasis objek yang dibuat oleh Microsoft dan dijalankan menggunakan .NET *framework*. C# merupakan salah satu bahasa pemrograman yang sering digunakan untuk *desktop application development*. Membuat aplikasi dengan C# dipermudah (terutama untuk membuat GUI) dengan adanya kakas yang bisa digunakan contohnya Windows Forms dalam Visual Studio.

Untuk membuat aplikasi C# memakai Windows Forms Visual Studio, langkah-langkahnya adalah sebagai berikut :

### 1. Buat *project* baru

- 1.1. Buka Visual Studio.
- 1.2. Pilih 'Create a new project'.
- 1.3. Pilih *template Windows Forms App (.NET Framework)* untuk C#.
- 1.4. Ketikkan nama *project*, lokasi penyimpanan, dan nama solusi yang diinginkan.

### 2. Membuat aplikasi

- 2.1. Klik tombol *Toolbox* di sebelah kiri, di bawah tombol *Data source* atau pilih menu *View > Toolbox*.
- 2.2. Pilih komponen yang ingin dimasukkan.
- 2.3. Akan tersedia menu *properties* di bawah kanan untuk mengubah komponen yang sudah dipilih.

2.4. Untuk menambahkan kode yang akan dijalankan saat komponen diklik atau interaksi lain, klik 2 kali pada komponen tersebut. Akan muncul tampilan untuk memasukkan kode yang diinginkan.

### **3. Menjalankan aplikasi**

3.1. Klik tombol Start di bagian atas. Akan muncul Windows Forms yang telah dibuat sebelumnya.

## **BAB III**

### **APLIKASI STRATEGI BFS/DFS**

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

Langkah awal dalam menyelesaikan persoalan *treasure hunt* adalah membagi persoalan ke dalam persoalan persoalan kecil untuk mempermudah dalam mencari solusi. Permasalahan utama pada tugas kali ini adalah menemukan rute pencarian menggunakan algoritma breadth-first search (BFS) dan algoritma depth-first search (DFS). Selain menemukan rute dengan dua algoritma tersebut, kita juga mendefinisikan permasalahan yang tidak kalah penting, yaitu menampilkan hasil pencarian dari kedua algoritma dalam sebuah aplikasi desktop.

Setelah mendefinisikan persoalan utama dalam menentukan solusi, langkah yang dikerjakan selanjutnya adalah mendesain pola berpikir dari algoritma breadth-first search (BFS) dan depth-first search (DFS) untuk menyelesaikan *treasure hunt*. Setelah berhasil menemukan pola algoritmanya, kelompok kami mulai menentukan apa saja yang diperlukan untuk mengimplementasikan algoritma tersebut, seperti library yang dipakai, tipe data yang akan dipakai, dan lain lain. Setelah memetakan apa saja yang diperlukan, kelompok kami mulai mengimplementasikan kode algoritma ke dalam bahasa C#.

Terakhir, setelah algoritma pencarian rute *treasure hunt* selesai, kami menyatukan algoritma yang sudah dibuat dengan GUI. GUI dibuat dengan memanfaatkan WinForms dalam Visual Studio.

## **3.2 Pemetaan Persoalan Menjadi Elemen BFS dan DFS**

### **3.2.1 Breadth First Search (BFS)**

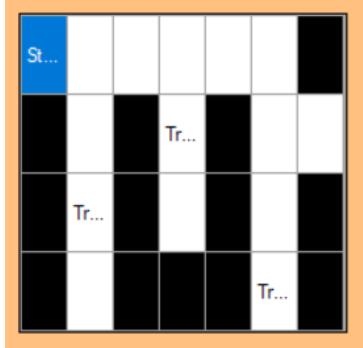
Penyelesaian dengan metode BFS memanfaatkan pencarian solusi memakai graf dinamis dan statis. Matriks pada peta dianggap mirip dengan sebuah graf, dengan isi simpul berupa K,R,X, atau T. Ketetanggan pada graf direpresentasikan dengan koordinat yang bersebelahan pada matriks. Untuk melakukan pencarian rutanya, digunakan graf dinamis yang direpresentasikan dengan *adjacency list* dalam bentuk *queue*. Pada setiap penelusuran, koordinat yang bertetanggan dengan koordinat yang dikunjungi saat ini akan dimasukkan ke dalam queue.

### **3.2.2 Depth First Search (DFS)**

Penyelesaian dengan DFS memanfaatkan konsep pencarian solusi memakai graf dinamis dan statis. Matriks peta dianggap mirip dengan sebuah graf, dengan isi simpul berupa K,R,X, atau T serta koordinatnya dan ketetanggan ditunjukkan dengan koordinat yang bersebelahan. Untuk pencarian rutanya sendiri menggunakan graf dinamis yang direpresentasikan dengan *adjacency list* dalam bentuk *stack*. *Stack* berisi koordinat yang dikunjungi. Digunakan dua *stack*, yang satu untuk menyimpan jalur solusi yang menuju ke harta karun, dan satu lagi untuk menyimpan keseluruhan jalur pencarian (untuk salah satu bonus).

## **3.3 Contoh Ilustrasi Kasus Lain**

Berikut adalah contoh kasus yang berbeda dengan yang ada di spesifikasi.



Dengan BFS, program akan memeriksa semua tetangga dari masing-masing simpul lebih dulu sebelum berpindah ke level berikutnya. Oleh karena itu, jalur pencarinya adalah sebagai berikut (direpresentasikan dengan koordinat (baris,kolom) yang titik 0,0-nya dimulai dari Start dan prioritas pencarian adalah atas bawah kiri kanan) :

(0,0) - (0,1) - (1,1) - (0,2) - (2,1) - (0,3) - (3,1) - (1,3) - (0,4) - (2,3) - (0,5) - (1,5) -  
(2,5) - (1,6) - (3,5) - (4,5)

Dari jalur pencarian di atas, didapat bahwa jalur untuk mendapat semua harta karun adalah sebagai berikut (ditandai dengan kotak hijau) :

*Congratulations! You found the treasure! :)*

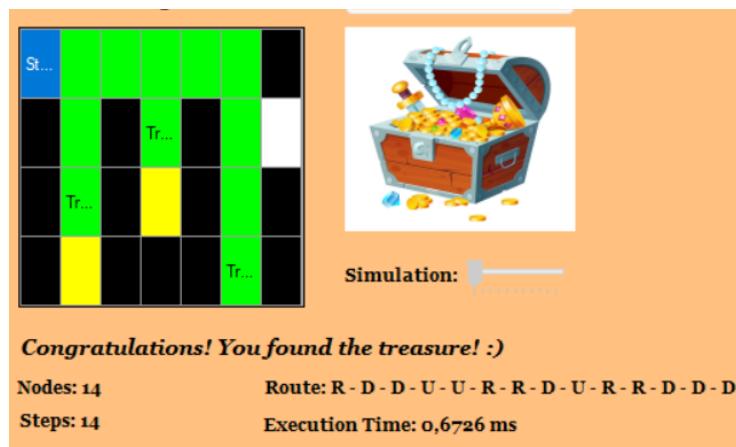
Nodes: 15      Route: R - D - D - U - U - R - R - D - U - R - R - D - D - D

Steps: 14      Execution Time: 3,1836 ms

Jalur pencarian akan berbeda untuk DFS, yang mengunjungi level yang lebih dalam lebih dulu. Jalur pencarian dengan DFS adalah sebagai berikut (prioritas arah masih atas bawah kiri kanan) :

(0,0) - (0,1) - (1,1) - (2,1) - (3,1) - (2,1) - (1,1) - (0,1) - (0,2) - (0,3) - (1,3) - (2,3) -  
(1,3) - (0,3) - (0,4) - (0,5) - (1,5) - (2,5) - (3,5) - (4,5)

Pencarian dihentikan saat mencapai koordinat (4,5) karena semua harta karun sudah ditemukan. Berbeda dengan BFS, pada DFS terjadi *backtrack* dalam pencarinya saat tidak ada tetangga lain yang belum dikunjungi. Dengan memanfaatkan sedikit heuristik (jalur *backtrack* hanya disimpan jika terjadi dari simpul harta karun), jalur hasil yang didapat adalah sebagai berikut :



## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi Program

Implementasi dari solusi ditulis dalam bentuk *pseudocode*. Implementasi hanya ditulis pada metode-metode yang penting dan tidak mencakup metode-metode yang berisi perhitungan teknis. Hal ini untuk menjaga *readability* dari *pseudocode* agar pembaca dapat memahami algoritma dengan baik.

##### 4.1.1 Implementasi Algoritma BFS

- Pada subbab ini, terdapat implementasi algoritma BFS.

```
procedure BFS(input maze : Matrix of char, input K : Tuple of int and int,
input tsp : bool, output path : list of tuple of int and int, output searchPath
: list of tuple of int and int)
{I.S maze, K, dan tsp terdefinisi, F.S : Jalur untuk mendapatkan semua
Treasure}

KAMUS
start, current : Tuple of int and int
queue = Queue of tuple
parent : Dictionary of Tuple
found : bool
temp, backHome : list of tuple

ALGORITMA PROSEDUR
start <- K
while(TreasureCount > 0) do
    searchPath.Add(start)
    queue.Enqueue(start)
    parent[start] <- null
    found <- false
    while(queue.Count > 0 and not(found)) do
```

```

        current = queue.Dequeue()

        if(maze[current.Item1, current.Item2] == 'T' and start != current) then
            temp <- BFS2point(maze,start, current)
            i traversal [0..temp.COUNT]
            path.Add(temp[i])
            start <- current
            TreasureCount <- TreasureCount -1
        if(current.Item1 > 0 and maze[current.Item1-1, current.Item2] != X and
not(parent.ContainsKey(current.Item1-1, current.Item2)))
            queue.Enqueue(current.Item1-1, current.Item2)
            temp <- (current.Item1-1, current.Item2)
            searchPath.Add(temp)
            parent[temp] <- current
        if(current.Item1 < maze.GetLength(0)-1 and maze[current.Item1+1,
current.Item2] != X and not(parent.ContainsKey(current.Item1+1,
current.Item2)))
            queue.Enqueue(current.Item1+1, current.Item2)
            temp <- (current.Item1+1, current.Item2)
            searchPath.Add(temp)
            parent[temp] <- current
        if(current.Item2 > 0 and maze[current.Item1, current.Item2-1] != X and
not(parent.ContainsKey(current.Item1, current.Item2-1)))
            queue.Enqueue(current.Item1, current.Item2-1)
            temp <- (current.Item1, current.Item2-1)
            searchPath.Add(temp)
            parent[temp] <- current
        if(current.Item2 < maze.GetLength(1)-1 and maze[current.Item1,
current.Item2+1] != X and not(parent.ContainsKey(current.Item1,
current.Item2+1)))
            queue.Enqueue(current.Item1, current.Item2+1)
            temp <- (current.Item1, current.Item2 +1)
            searchPath.Add(temp)
            parent[temp] <- current

        if(maze[current.Item1, current.Item2] == 'T' and start != current) then
            found <- true
    if(tsp) then

```

```

backHome <- BFS2point(maze, start, K)
for i in traversal[0..backHome.Count]
    path.Add(backHome[i])
for i in traversal[0..path.Count-1]
    if(path[i] == path[i+1] then
        path.RemoveAt(i)

```

```

function BFS2point(Matrix of char maze, Tuple of int and int titik1, Tuple of int and int titik2) -> List of Tuple of int and int
{Mengembalikan list yang berisi koordinat koordinat yang merupakan jalur dari koordinat titik1 sampai ke titik2}

KAMUS
current : Tuple of int and int
queue = Queue of tuple
parent : Dictionary of Tuple
path, kosong : list of tuple

ALGORITMA FUNGSI
queue.Enqueue(titik1)
parent[titik1]<- null
while(queue.Count > 0)
    current <- queue.Dequeue()
    if(current = titik2) then
        while(current != null) do
            path.Add(current)
            current <- parent[current]
        path.Reverse()
        -> path
    if(current.Item1 > 0 and maze[current.Item1-1, current.Item2] != X and
not(parent.ContainsKey(current.Item1-1, current.Item2)))then
        queue.Enqueue(current.Item1-1, current.Item2)
        temp <- (current.Item1-1, current.Item2)
        parent[temp] <- current
    if(current.Item1 < maze.GetLength(0)-1 and maze[current.Item1+1,

```

```

current.Item2] != X and not(parent.ContainsKey(current.Item1+1,
current.Item2))) then
    queue.Enqueue(current.Item1+1, current.Item2)
    temp <- (current.Item1+1, current.Item2)
    parent[temp] <- current
    if(current.Item2 > 0 and maze[current.Item1, current.Item2-1] != X and
not(parent.ContainsKey(current.Item1, current.Item2-1))) then
        queue.Enqueue(current.Item1, current.Item2-1)
        temp <- (current.Item1, current.Item2-1)
        parent[temp] <- current
        if(current.Item2 < maze.GetLength(1)-1 and maze[current.Item1,
current.Item2+1] != X and not(parent.ContainsKey(current.Item1,
current.Item2+1))) then
            queue.Enqueue(current.Item1, current.Item2+1)
            temp <- (current.Item1, current.Item2 +1)
            parent[temp] <- current
    kosong <- {}
-> kosong

```

#### 4.1.2 Implementasi Algoritma DFS

Pada subbab ini, terdapat implementasi algoritma DFS.

```

procedure DFS(input m : Matrix of char, input tsp : boolean, output path : List
of Tuple of integer, input totalTreasure, startRow, startCol : integer)
{I.S. Boolean tsp dan matrix m sudah terdefinisi.
 F.S. Jalur hasil pencarian yang mendapat semua harta karun dan jalur kembali
ke titik awal (jika tsp bernilai true) dikembalikan}
KAMUS LOKAL
treasureCount : integer
i,j : integer
stuck: boolean
procedure visitAdjacent(input/output i,j : integer, output isStuck, input
maxVisit)
{Mengunjungi simpul tetangga yang valid (bukan tembok, masih dalam batas
matrix, dan belum pernah dikunjungi sebelumnya atau jumlah dikunjungi <
maxVisit) dengan prioritas arah atas,bawah,kiri,kanan.

```

```

I.S. i,j terdefinisi, isStuck sembarang
F.S. i,j diubah ke koordinat simpul tetangga valid atau tetap sama, isStuck bernilai false jika ada tetangga yang bisa dikunjungi atau true jika tidak

procedure insertLast(input/output path : List of Tuple of integer, input p : Tuple of integer)
{I.S. path sembarang, p terdefinisi dan berisi titik yang ingin dimasukkan.
 F.S. p dimasukkan sebagai elemen terakhir p}

procedure deleteLast(input/output path : List of Tuple of integer)
{I.S. path tidak kosong.
 F.S. elemen terakhir path dihapus}

ALGORITMA PROSEDUR

treasureCount <- 0
i <- startRow; j <- startCol
maxVisit <- 1

while (treasureCount < totalTreasure) do
    visitAdjacent(i,j,stuck,maxVisit)
    if (not stuck) then
        insertLast(path, <i,j>)
        if (m[i,j] = 'T') then
            treasureCount <- treasureCount + 1
        else
            if (m[i,j] = 'T') then
                {Simpan jalur backtrack karena treasure tetap harus diambil}
                insertLast(path, elemen kedua terakhir path)
                maxVisit <- maxVisit + 1
            else
                deleteLast(path)
                <i,j> = path[Last]
    if (tsp) then
        while (i ≠ startRow or j ≠ startCol) do
            visitAdjacent(i,j,stuck,maxVisit)
            if (not stuck) then
                insertLast(path, <i,j>)
            else
                deleteLast(path)
                <i,j> = path[Last]

```

```

procedure visitAdjacent(input/output i,j : integer, output isStuck, input
maxVisit)
{Mengunjungi simpul tetangga yang valid (bukan tembok, masih dalam batas
matrix, dan belum pernah dikunjungi sebelumnya atau jumlah dikunjungi <
maxVisit) dengan prioritas arah atas,bawah,kiri,kanan.
I.S. i,j terdefinisi, isStuck sembarang
F.S. i,j diubah ke koordinat simpul tetangga valid atau tetap sama, isStuck
bernilai false jika ada tetangga yang bisa dikunjungi atau true jika tidak}

KAMUS LOKAL

ALGORITMA PROSEDUR

isStuck <- false
if (atas valid dan belum dikunjungi) then
    i <- i-1
else if (bawah valid dan belum dikunjungi) then
    i <- i+1
else if (kiri valid dan belum dikunjungi) then
    j <- j-1
else if (kanan valid dan belum dikunjungi) then
    j <- j+1
else if (atas valid dan visit atas < maxVisit) then
    i <- i-1
else if (bawah valid dan visit bawah < maxVisit) then
    i <- i+1
else if (kiri valid dan visit kiri < maxVisit) then
    j <- j-1
else if (kanan valid dan visit kanan < maxVisit) then
    j <- j+1
else
    isStuck <- true

```

## 4.2 Penjelasan Struktur Data dan Spesifikasi Program

Dalam program ini digunakan beberapa jenis struktur data, yaitu :

- Matriks

Struktur matriks diimplementasikan sebagai sebuah *class* dengan atribut berupa :

- *Array of char*
- Koordinat titik mulai (indeks yang berisi huruf 'K')
- Jumlah baris dan kolom
- Jumlah harta karun

b. *Tuple*

*Tuple* menggunakan implementasi dari kelas bawaan System.Collections.Generic. *Tuple* dipakai untuk menyimpan koordinat yang dikunjungi saat memakai algoritma DFS maupun BFS.

c. *List*

*List* menggunakan implementasi dari kelas bawaan System.Collections.Generic. *List* digunakan untuk menyimpan *tuple* koordinat yang telah dikunjungi dalam algoritma BFS maupun DFS.

*Method* yang digunakan adalah :

- *Add* : memasukkan elemen di akhir *list*.
- *Insert* : memasukkan elemen di indeks yang diinginkan.

d. *Stack*

*Stack* diimplementasikan menggunakan *collection* yang ada dalam kelas bawaan System.Collections.Generic. *Stack* digunakan untuk menyimpan *tuple* yang berisi titik koordinat yang dikunjungi saat memakai algoritma DFS. *Method* yang digunakan adalah :

- *Push* : menambahkan elemen baru ke *stack* dengan aturan LIFO (*Last In First Out*).
- *Pop* : mengambil sekaligus menghapus elemen terakhir yang dimasukkan ke *stack*.
- *Peek* : melihat elemen yang terakhir dimasukkan ke *stack* tanpa menghapus elemen tersebut.

#### e. *Queue*

*Queue* diimplementasikan menggunakan *collection* yang ada dalam kelas bawaan System.Collections.Generic. *Queue* digunakan untuk menyimpan tuple yang berisi koordinat yang dikunjungi saat memakai algoritma BFS. *Method* yang digunakan adalah :

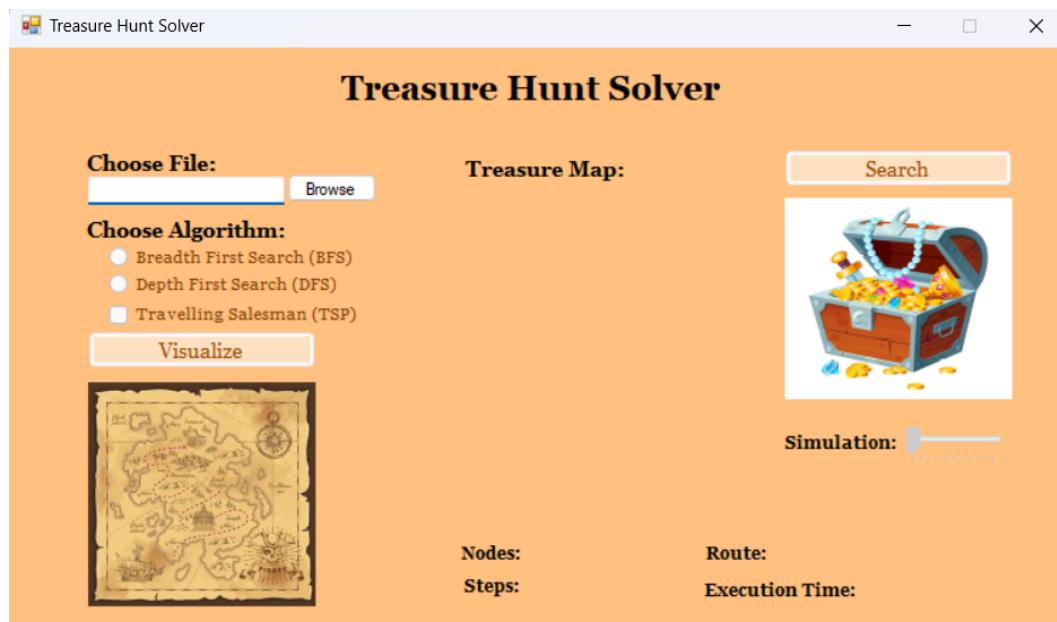
- *Enqueue* : memasukkan elemen baru kedalam *queue* dengan aturan FIFO (*First In First Out*).
- *Dequeue* : mengambil dan menghapus elemen pertama yang dimasukkan kedalam *queue*.

#### f. *Dictionary*

*Dictionary* diimplementasikan menggunakan *collection* yang ada dalam kelas bawaan System.Collections.Generic. *Dictionary* digunakan untuk menyimpan koordinat yang menurunkan koordinat yang sekarang sedang dikunjungi. Hal ini digunakan untuk menentukan jalur pencarian ketika sudah bertemu dengan *treasure*.

### 4.3 Tata Cara Penggunaan Program

Untuk menjalankan program, cukup jalankan file "Treasure Hunt Solver.exe" pada **folder bin**. Berikutnya, pilih file yang berisi peta yang ingin divisualisasikan dengan menekan tombol "Browse". Setelah memilih file, tekan tombol BFS atau DFS untuk memilih algoritma yang diinginkan dan tombol TSP apabila ingin mendapat solusi yang kembali ke titik awal. Selanjutnya klik tombol "Visualize" untuk menampilkan peta. Apabila ingin melihat solusi, tekan tombol "Search". Akan muncul jalur pencarian yang dilakukan dan jalur hasil setelah pencarian selesai. Berikut adalah tampilan program yang kami buat.



### 4.4 Hasil Pengujian dan Analisis

Berikut adalah hasil uji dari program Treasure Hunt kami. Kotak kuning menandakan simpul yang dikunjungi, dan kotak hijau menandakan jalur hasil.

- Test Case 1

Isi file :

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

Hasil :



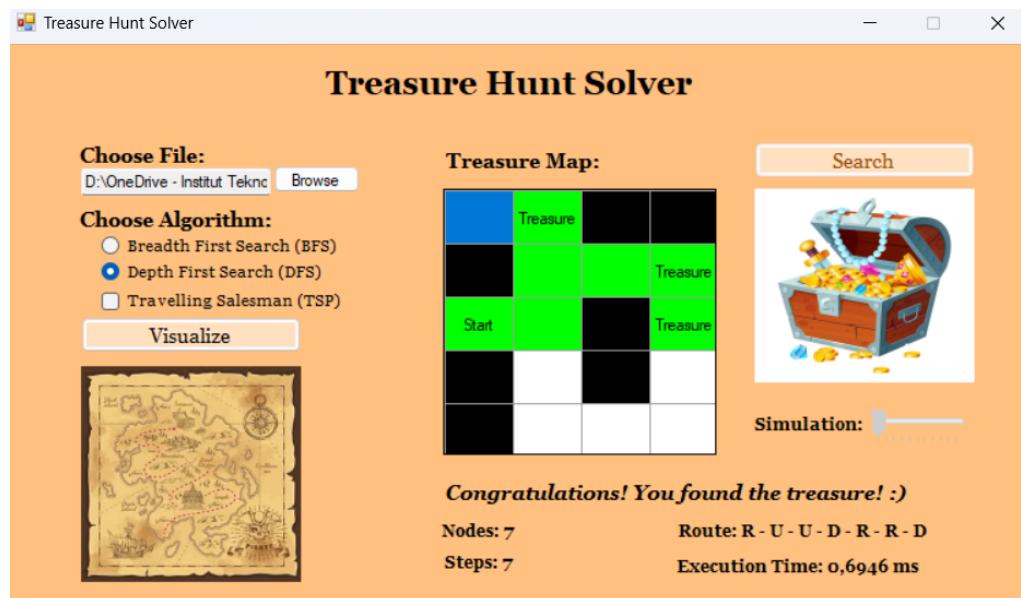
Visualisasi



### BFS Non-TSP



### BFS TSP



### DFS Non-TSP



### DFS TSP

Analisis :

Untuk pencarian non-TSP, bisa dilihat bahwa algoritma DFS yang digunakan lebih baik karena tidak mengunjungi semua simpul dan waktu eksekusinya lebih pendek. Hal ini karena BFS memang memeriksa semua tetangga lebih dulu ketimbang memeriksa simpul yang lebih dalam. Hasil DFS ini juga dipengaruhi prioritas arah yang dipilih. Kebetulan dengan peta seperti di atas, DFS tidak perlu menelusuri seluruh matriks lebih dulu sebelum menemukan semua harta karun.

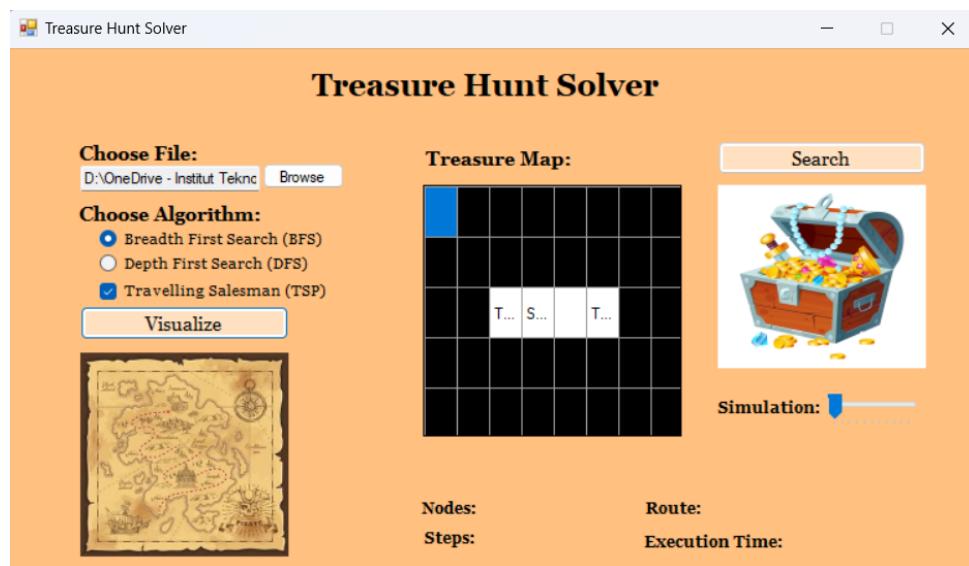
Untuk pencarian TSP, algoritma BFS lebih baik karena jumlah langkahnya lebih sedikit. Hal ini karena pada TSP dengan DFS, program memprioritaskan mengunjungi simpul yang belum pernah dikunjungi sebelumnya (dalam kasus ini simpul di bawah harta karun terakhir).

- Test case 2

Isi file :



Hasil :



Visualisasi



BFS Non-TSP



BFS TSP



DFS Non-TSP



DFS TSP

Analisis :

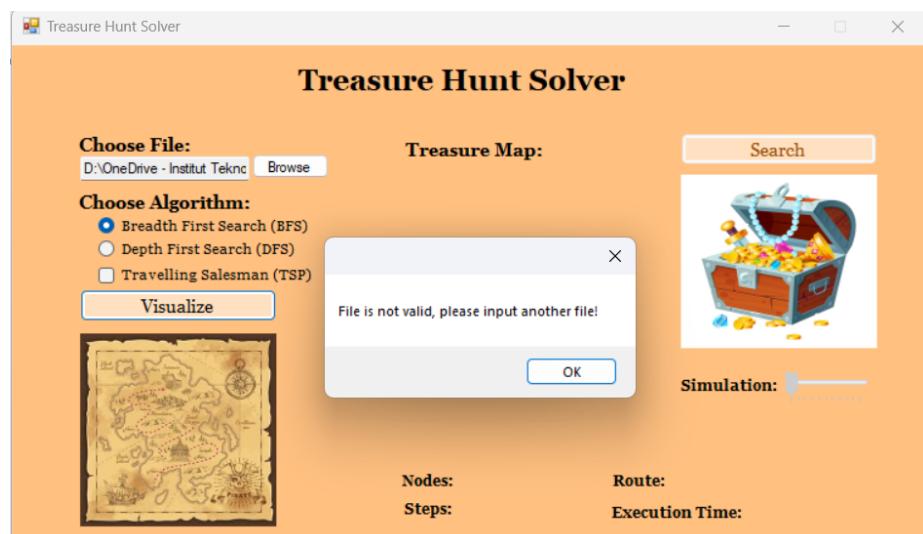
Hasil dari kedua algoritma sama untuk TSP maupun Non-TSP, dengan DFS lebih cepat dibanding BFS untuk Non-TSP. Hal ini dikarenakan pergerakan yang bisa dilakukan terbatas pada kiri dan kanan saja, sehingga tidak ada banyak alternatif jalur yang bisa diambil.

- Test Case 3

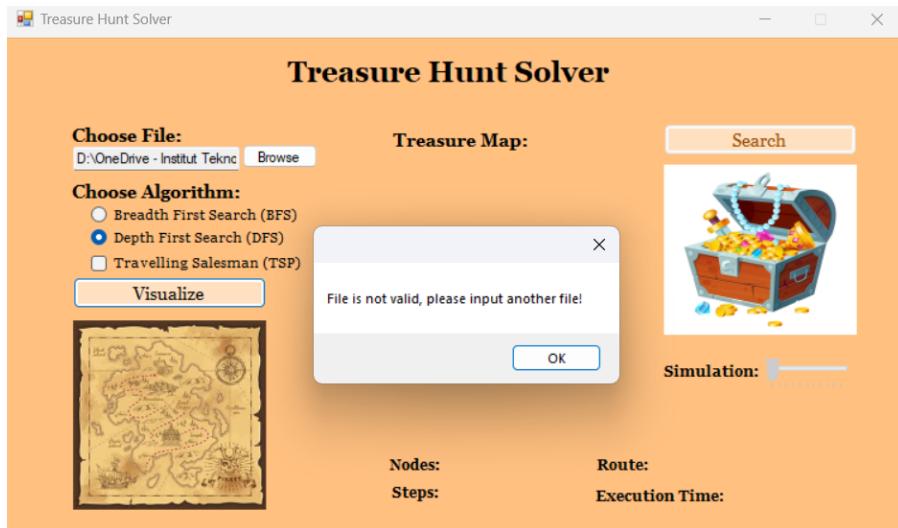
Isi file :

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

Hasil :



BFS



DFS

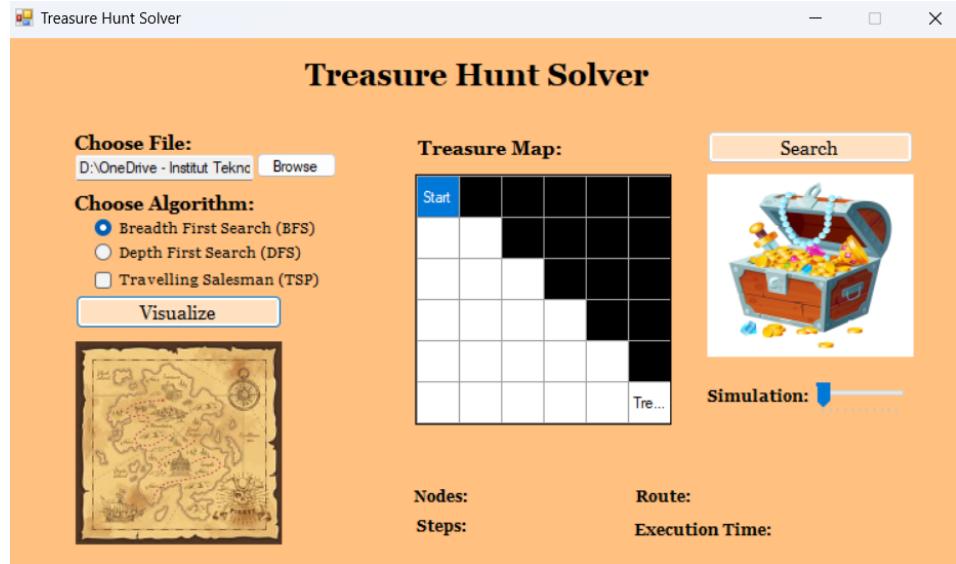
Analisis : Isi file tidak valid karena tidak hanya terdiri dari 'K', 'R', dan 'T'.

- Test Case 4

Isi file :

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

Hasil :



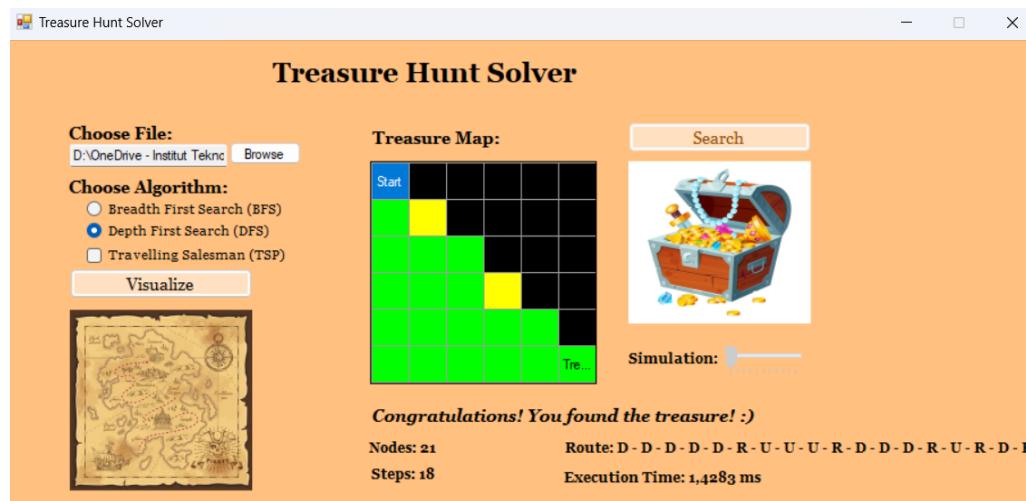
Visualisasi



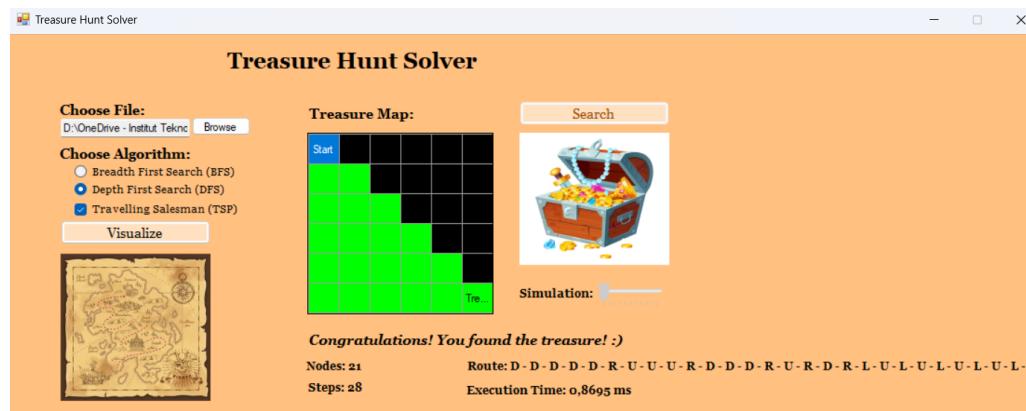
BFS Non-TSP



BFS TSP



DFS Non-TSP



DFS TSP

Analisis :

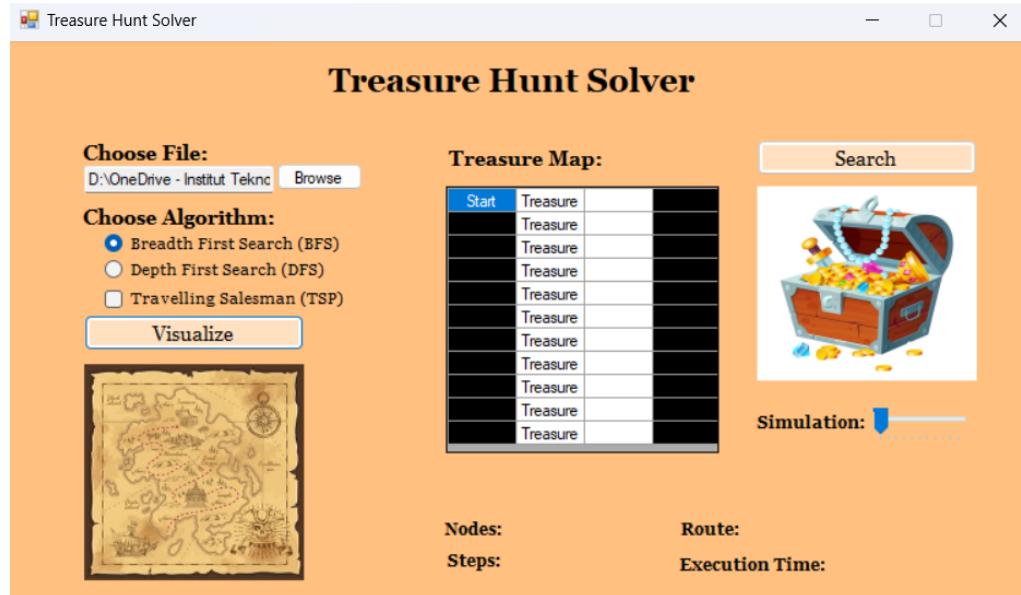
Jalur hasil BFS jauh lebih baik dibanding DFS untuk TSP maupun Non-TSP. Hal ini karena dengan prioritas arah yang diambil, DFS perlu menelusuri semua simpul lebih dulu sebelum mendapat harta karun, sedangkan BFS menemukan jalur yang lebih pendek karena menguji semua tetangga dari suatu simpul lebih dulu. Untuk jalur kembalinya keduanya memiliki hasil yang sama, dipengaruhi oleh prioritas arah yang dipilih juga.

- Test Case 5

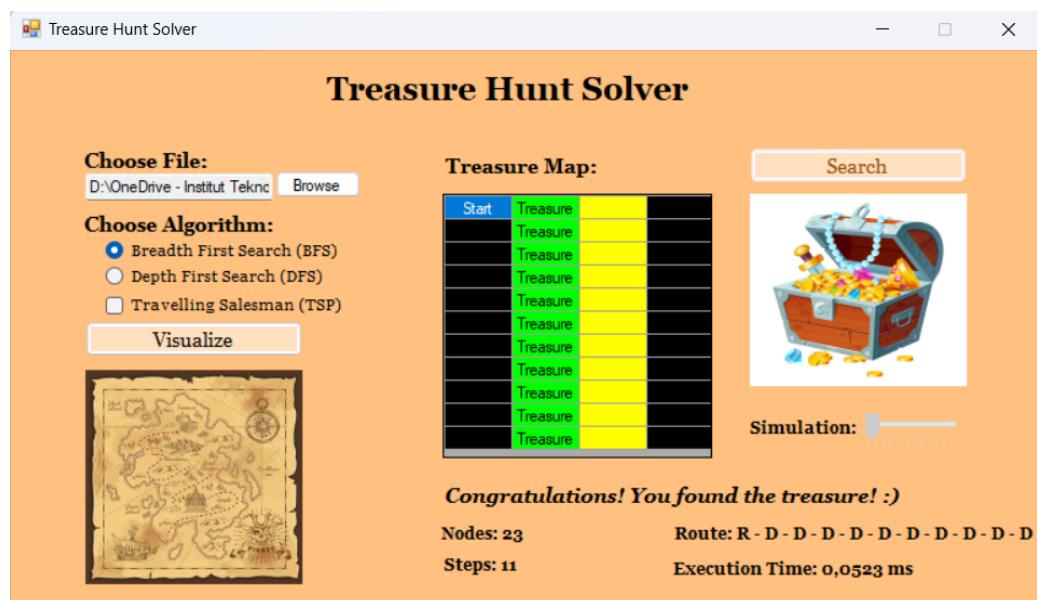
Isi *file* :

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

Hasil :



## Visualisasi



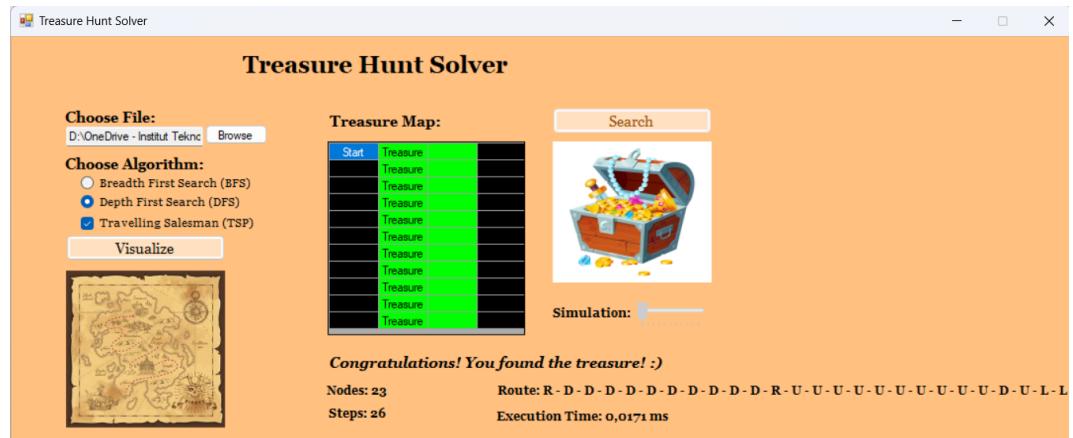
BFS Non-TSP



BFS TSP



DFS Non-TSP



## DFS TSP

Analisis :

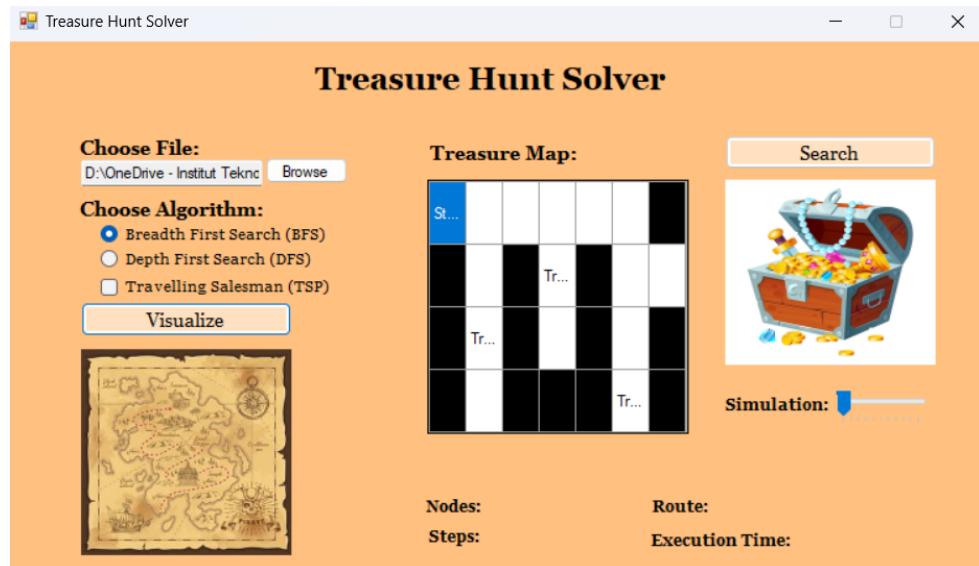
Untuk pencarian non-TSP, DFS dan BFS memiliki hasil yang sama, dengan DFS lebih cepat dibanding BFS. Hal ini karena pada DFS, tidak semua simpul perlu dikunjungi untuk mendapat semua harta karun karena semuanya terletak menurun dan arah bawah merupakan prioritas kedua setelah. Namun untuk TSP, BFS lebih baik daripada DFS karena DFS menelusuri simpul yang belum pernah dikunjungi lebih dulu.

- Test Case 6 :

Isi file :

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

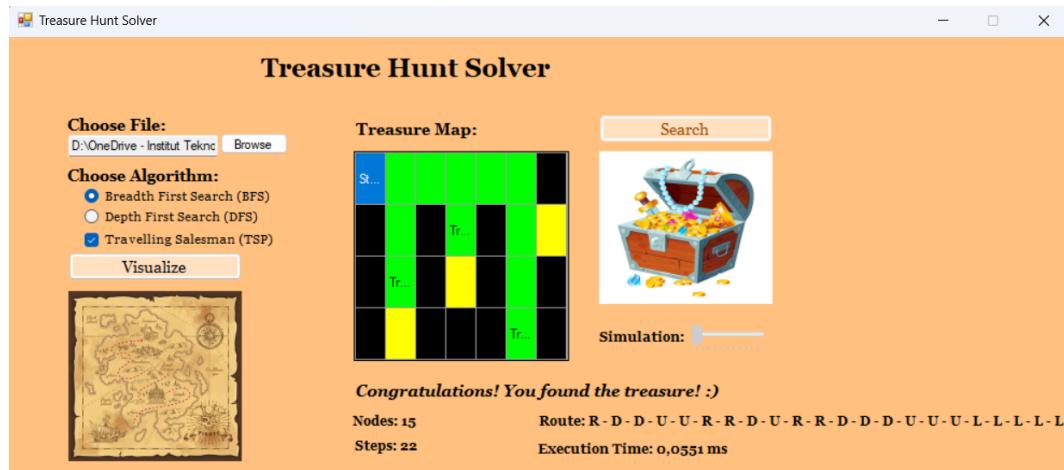
Hasil :



Visualisasi



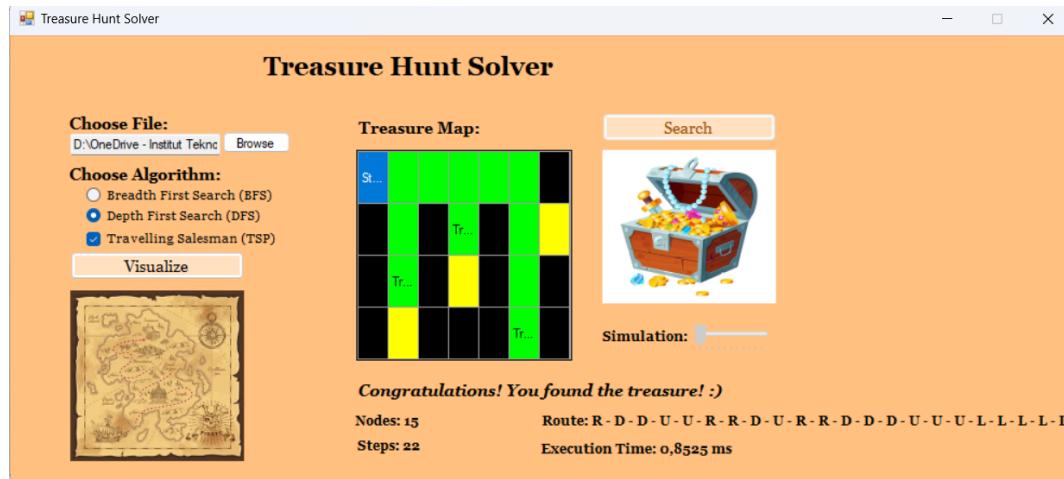
BFS Non-TSP



BFS TSP



DFS Non-TSP



### DFS TSP

Analisis :

Hasil dari kedua algoritma relatif sama dengan DFS lebih cepat dibanding BFS. Hal ini karena pada peta di atas, pergerakan yang bisa dilakukan relatif terbatas sehingga tidak banyak alternatif jalur yang bisa diambil.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Dari pengerjaan tugas besar II IF2211 Strategi Algoritma ini, kami memperoleh kesimpulan-kesimpulan sebagai berikut.

- Kami berhasil mengimplementasikan algoritma *Breadth First Search* (BFS) dalam permasalahan pencarian *treasure* dalam suatu *maze*.
- Kami berhasil mengimplementasikan algoritma *Depth First Search* (DFS) dalam permasalahan pencarian *treasure* dalam suatu *maze*.
- Kami berhasil membuat GUI untuk memvisualisasikan pencarian rute untuk mendapatkan *treasure* dalam *maze*.

#### **5.2 Saran**

Berdasarkan pengalaman mengerjakan Tugas Besar ini, kelompok kami menyarankan apabila persoalan persoalan sudah selesai dikerjakan sesuai dengan tugasnya, alangkah lebih baik jika mengerjakan bersama sama secara *real-time* dalam menyatukan komponen komponen persoalan. Hal ini dilakukan agar tidak terjadi kebingungan.

#### **5.3 Refleksi**

Walaupun Tugas Besar ini bertabrakan dengan UTS maupun *deadline* Tugas besar lainnya, kelompok kami berhasil menyelesaikan tugas besar ini dengan aman dan terkendali.

#### **5.4 Tanggapan**

Tugasnya sangat mantap dan menambah wawasan tentang DFS/BFS.

## **DAFTAR PUSTAKA**

Munir, R. dan Nur Ulfa Maulidevi (2023). Breadth/Depth First Search (BFS/DFS) (Bagian 1). IF2211 Strategi Algoritma - Semester II Tahun 2022/2023.

Diakses pada 22 Maret 2023, pukul 10.31 dari

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

Microsoft. “Create a Windows Forms App in Visual Studio with C#”,

<https://learn.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022>, diakses pada 22 Maret 2023.

# **LAMPIRAN**

## **Link Repository Program**

[https://github.com/Ulung32/Tubes2\\_NUB](https://github.com/Ulung32/Tubes2_NUB)

## **Link Video Demo Youtube**

<https://youtu.be/Nw8Y4F5S9dw>