IF2211
Strategi Algoritma

LAPORAN TUGAS BESAR

STRATEGI ALGORITMA

PENGAPLIKASIAN ALGORITMA BFS DAN DFS DALAM MENYELESAIKAN PERSOALAN MAZE TREASURE HUNT

Kelas Mahasiswa K03

Dosen Pengampu: Ir. Rila Mandala, M.Eng., Ph.D.

Diajukan sebagai tugas besar Mata Kuliah IF2211 Strategi Algoritma pada

Semester II Tahun Akademik 2022/2023

Disusun Oleh:

Kelompok 18 (xixi)

Jason Rivalino	**13521008**
Syarifa Dwi Purnamasari	**13521018**
Agsha Athalla Nurkareem	**13521027**

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

IF2211
Strategi Algoritma

DAFTAR ISI

DAFTAR ISI 2

BAB I: DESKRIPSI TUGAS 3

BAB II: LANDASAN TEORI 5

I. Teori Dasar Graph Traversal, Algoritma BFS dan DFS 5

II. Penjelasan C# Desktop Application Development 7

BAB III: ANALISIS PEMECAHAN MASALAH 10

I. Langkah-langkah Pemecahan Masalah 10

II. Elemen-elemen Algoritma BFS dan DFS 10

III. Ilustrasi Kasus Lain 11

BAB IV: IMPLEMENTASI DAN PENGUJIAN 12

I. Implementasi Program 12

II. Struktur Data dan Spesifikasi Program 22

III. Tata Cara Penggunaan Program 22

IV. Hasil Pengujian 26

V. Analisis Desain Solusi Algoritma BFS dan DFS 32

BAB V: KESIMPULAN DAN SARAN 34

I. Kesimpulan 34

II. Saran 34

III. Refleksi 34

IV. Tanggapan 35

DAFTAR PUSTAKA 36

LAMPIRAN 37

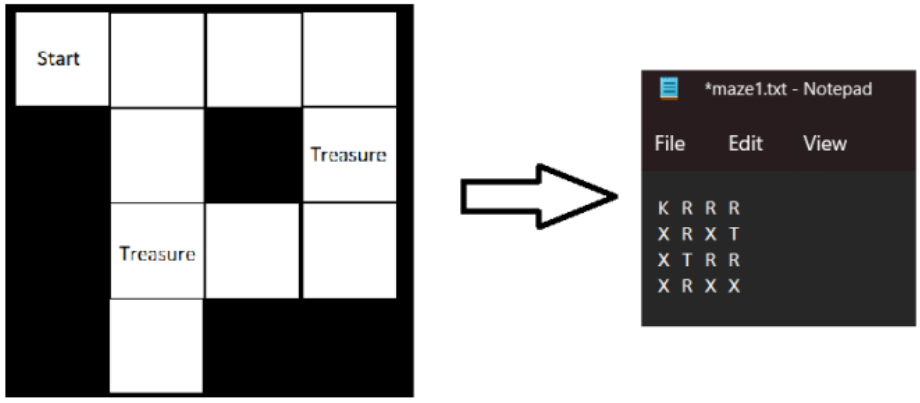
BAB I

DESKRIPSI TUGAS

Dalam tugas besar ini, tugas yang diberikan adalah 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 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, 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.

Daftar input maze akan dikemas dalam sebuah folder yang dinamakan test dan terkandung dalam repository program. Folder tersebut akan setara kedudukannya dengan folder src dan doc (struktur folder repository akan dijelaskan lebih lanjut di bagian bawah spesifikasi tubes). Cara input maze boleh langsung input file atau dengan textfield sehingga pengguna dapat mengetik nama maze yang diinginkan. Apabila dengan textfield, harus menghandle kasus apabila tidak ditemukan dengan nama file tersebut.

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

BAB II

LANDASAN TEORI

I. Teori Dasar Graph Traversal, Algoritma BFS dan DFS

Graph Traversal adalah algoritma yang mengunjungi simpul-simpul dengan cara yang sistematik. Graph traversal merupakan proses pencarian solusi dari representasi persoalan dalam graf. Algoritma ini sendiri dapat dibagi dua berdasarkan jenis pencarian solusinya yaitu:

1) Algoritma Pencarian Melebar (*Breadth First Search atau BFS*)

Algoritma BFS adalah algoritma yang melakukan pencarian dari satu simpul ke simpul lainnya secara melebar. Untuk bentuk graf dengan representasi G = (V, E), tahapan algoritma yang ada dalam proses pencarian ini yaitu:

a) Memulai penelusuran dari simpul v
b) Mengunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu
c) Melakukan penelusuran terhadap simpul-simpul yang belum dikunjungi dan bertetangga dengan simpul yang sebelumnya dikunjungi. Tahapan ini dilakukan hingga semua simpul dikunjungi atau simpul yang dicari telah ditemukan

Adapun struktur data yang ada dalam algoritma DFS ini antara lain:

a) Matriks ketetanggaan A = [a_{ij}] dengan ukuran n x n, a_{ij} akan bernilai 1 jika simpul i dan j bertetangga dan bernilai 0 jika simpul i dan j tidak bertetangga
b) Antrian (*Queue)* untuk menyimpan simpul yang telah dikunjungi
c) Tabel Boolean untuk menentukan apakah simpul i sudah dikunjungi atau belum

Untuk *pseudocode* dari Algoritma BFS sendiri adalah sebagai berikut:

```
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 : antrean
```

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

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

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

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

Algoritma
    BuatAntrean(q) { buat antrian kosong }

    write(v) { cetak simpul awal yang dikunjungi }
    dikunjungi[v] <- true { simpul v telah dikunjungi,
                            tandai dengan true }

    MasukAntrean(q,v) { masukkan simpul awal kunjungan ke
                        dalam antrian }

    { kunjungi semua simpul graf selama antrean belum kosong }

    while not AntrianKosong(q) do
         HapusAntrean(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}
                        MasukAntrean(q,w)
                        dikunjungi[w] <- true
                endif
         endfor
    endwhile
    { AntrianKosong(q) }
```

2) Algoritma Pencarian Mendalam (*Depth First Search atau DFS*)

Algoritma DFS adalah algoritma yang melakukan pencarian dari satu simpul ke simpul lainnya secara mendalam. Adapun tahapan algoritma yang ada dalam proses pencarian ini yaitu:

a) Memulai penelusuran dari simpul v
b) Mengunjungi simpul w yang bertetangga dengan simpul v
c) Mengulangi DFS dari simpul w
d) Melakukan pencarian runut-balik (*backtracking*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi ketika sudah mencapai simpul u dan semua simpul yang bertetangga dengan simpul u sudah dikunjungi
e) Pencarian berakhir ketika tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi

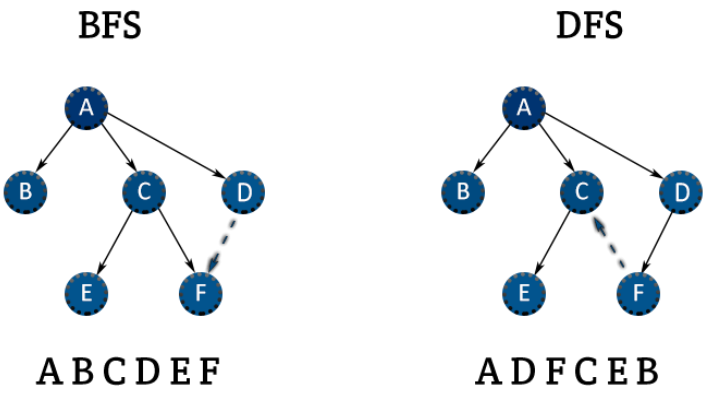
Untuk *pseudocode* dari Algoritma DFS sendiri adalah sebagai berikut:

```
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
```

Perbandingan antara algoritma BFS dan DFS dalam melakukan pencarian dapat diilustrasikan dengan gambar berikut:



Sumber: https://www.boardinfinity.com/blog/difference-between-bfs-and-dfs/

II. Penjelasan C# Desktop Application Development

C# Desktop Application Development adalah kumpulan perlengkapan *software* yang dapat dipergunakan untuk membangun aplikasi *desktop*. Bahasa pemrograman yang dipergunakan dalam *software* ini adalah C# (C-Sharp). Dalam pengembangan aplikasi menggunakan C# Desktop Application Development, terdapat

berbagai macam *tools* dan *framework* yang dapat dipergunakan seperti .NET Framework, Windows Forms, dan WPF dengan bantuan IDE Visual Studio. *Tools-tools* ini dapat dipergunakan untuk membantu membuat tampilan antarmuka (UI) aplikasi menjadi lebih menarik.

Komponen-komponen penting yang ada dalam C# Desktop Application Development sendiri antara lain:

1) Visual Studio

   Visual Studio atau Microsoft Visual Studio adalah adalah *software* yang dapat digunakan untuk mengembangkan aplikasi. Visual Studio mencakup SDK, IDE, Compiler, dan berbagai macam komponen lainnya untuk membantu mengembangkan aplikasi dalam *native code* (bahasa mesin yang berjalan diatas Windows) ataupun *managed code* (dalam bentuk .NET Framework). Aplikasi ini dapat digunakan untuk membuat program aplikasi, objek-objek pembantu program, dan proses *debugging*.

2) .NET Framework

   .NET Framework adalah *software* yang menyediakan berbagai macam layanan untuk aplikasi yang sedang berjalan di Windows. *Software* ini menyediakan dua komponen utama yaitu Common Language Runtime (CLR) yang berfungsi untuk menangani aplikasi yang sedang berjalan dan Framework Class Library (FCL) yang merupakan pustaka kode terkelola untuk melakukan pemrograman pada aplikasi. Adapun fungsi dari .NET Framework sendiri antara lain untuk menyediakan lingkungan pemrograman yang konsisten, menyediakan lingkungan eksekusi kode yang tepat, dan mengintegrasikan kode satu dengan kode lainnya.

3) Windows Forms

   Windows Forms adalah *Class Library* yang berfungsi untuk mempermudah pengembangan aplikasi *desktop*. Aplikasi ini bersifat GUI dan tergabung dalam .NET Framework. Dalam Windows Forms, terdapat berbagai macam variasi tools untuk membantu memudahkan pengembangan aplikasi berbasis Windows.

Untuk langkah-langkah dalam mengembangkan aplikasi *desktop* dengan Windows Form App adalah sebagai berikut:

1) Membuat *Project*

   Pembuatan *project* baru dalam aplikasi ini dimulai dengan membuka aplikasi Visual Studio 2022, lalu memilih *Create a new project*, pilih menu untuk membuat *Windows Form App (.NET Framework)*, lalu mengisi nama *project* dan menekan tombol *Create* untuk membuat *project* baru.

2) Membuat Aplikasi

   Pembuatan aplikasi dilakukan dengan memilih menu *toolbox* dan memilih komponen-komponen yang ingin digunakan dalam tampilan aplikasi yang ingin dibuat. Fitur-fitur yang ada dalam *toolbox* antara lain label, textbox, button, datagridview, time, dll. Jika ingin dimodifikasi, bisa diubah propertinya pada bagian *properties*.

3) Menambahkan Kode

   Pengeditan pada aplikasi dapat dilakukan pada file Form1.cs yang dapat diakses dengan menekan dua kali pada file Form1.cs [Design].

4) Menjalankan aplikasi

   Untuk menjalankan aplikasi, cukup menekan tombol *Start* pada menu.

BAB III

ANALISIS PEMECAHAN MASALAH

I. Langkah-langkah Pemecahan Masalah

Langkah-langkah pemecahan permasalahan yang kelompok kami terapkan dalam penyelesaian tugas besar ini antara lain:

1) Memahami proses penyelesaian *maze* untuk memperoleh semua harta karun yang ada
2) Memecah solusi penyelesaian yaitu dengan Algoritma BFS dan DFS
3) Menyusun perancangan algoritma BFS dan DFS untuk mencari jarak yang terdekat dalam memperoleh semua harta karun yang ada
4) Menyusun GUI program dengan Windows Forms yang terdapat dalam Visual Studio
5) Menggunakan *toolbox* datagridview untuk memproyeksikan bentuk *maze* dari file
6) Menggabungkan antara program dengan GUI yang telah dibuat

II. Elemen-elemen Algoritma BFS dan DFS

Untuk elemen-elemen yang terdapat pada Algoritma BFS dan DFS antara lain:

Elemen	Representasi
Nodes	Semua *grid* dari *maze* yang ditelusuri untuk mencari keseluruhan harta karun
Edges	Proses perpindahan dari satu *grid* pada *maze* ke *grid* lainnya
Goals	Mendapatkan semua harta karun yang ingin dicari

Untuk permasalahan yang ada pada algoritma BFS, permasalahan diselesaikan dengan menggunakan sistem antrian (*Queue*) yaitu menyimpan *node* yang baru dicek pada antrian belakang (*enqueue*) dan membuang *node* yang ada pada antrian depan (*dequeue*) sesuai dengan prinsip *Queue* yaitu *First In First Out (FIFO)*.

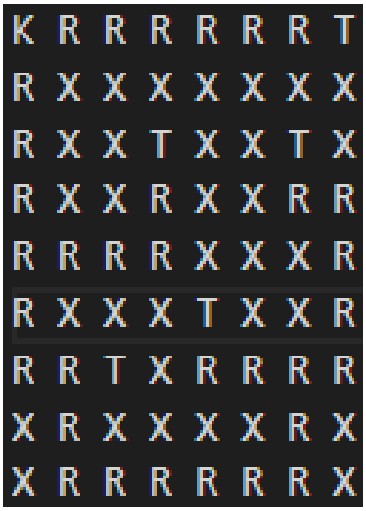
Untuk permasalahan yang ada pada algoritma DFS, permasalahan diselesaikan dengan menggunakan sistem tumpukan (*Stack*) yaitu menyimpan *node* yang baru dicek

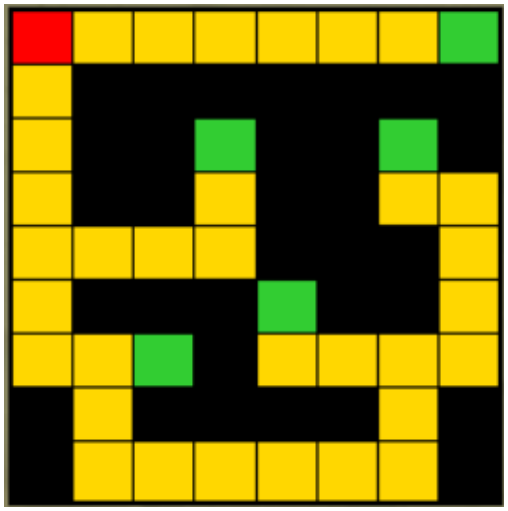
pada tumpukan teratas (*push*) dan membuang *node* yang ada pada tumpukan teratas juga (*pop*) sesuai dengan prinsip *Stack* yaitu *Last In First Out (LIFO).*

III. Ilustrasi Kasus Lain

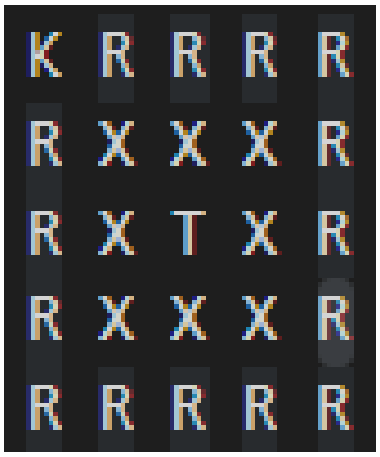
Beberapa contoh ilustrasi kasus lain yang mungkin terjadi dalam pencarian harta karun di dalam *maze* antara lain sebagai berikut:

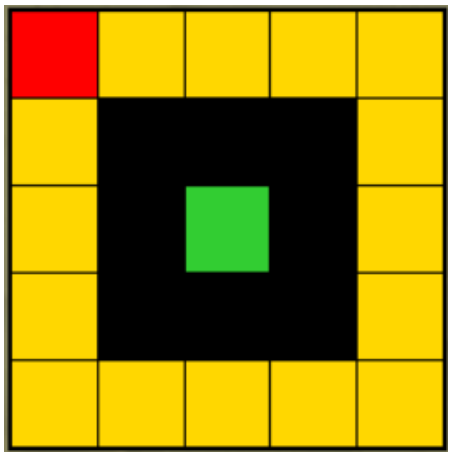
1) Pencarian treasure hanya bisa dilakukan secara *backtracking*





2) Kondisi terdapat harta karun yang tidak bisa diambil karena terkurung oleh *grid* dalam *maze*. Pada kasus ini, algoritma kami tidak dapat meng-*handle case* ini karena algoritma kami akan terus melakukan looping hingga semua node treasure dilewati, sehingga pada *case* ini, program kami akan melakukan *endless looping*.





BAB IV

IMPLEMENTASI DAN PENGUJIAN

I. Implementasi Program

1) Pseudocode untuk algoritma BFS

```
class cariBFS:

    function checkAdjacent(x1, y1, x2, y2):
        if x1 = x2 + 1 AND y1 = y2:
            -> "U"
        else if x1 = x2 - 1 AND y1 = y2:
            -> "D"
        else if x1 = x2 AND y1 = y2 + 1:
            -> "L"
        else if x1 = x2 AND y1 = y2 - 1:
            -> "R"
        -> ""

    function printStep(array):
        step <- ""
        for i <- 0 to (array.Length / 2) - 1:
            step += checkAdjacent(array[i, 0], array[i, 1], array[i + 1, 0],
array[i + 1, 1])
        -> step

    function cariPlayer(map, baris, kolom):
        lokasiPlayer <- [0, 0]
        for k <- 0 to baris - 1:
            for l <- 0 to kolom - 1:
                if map[k, l] = 'K':
                    lokasiPlayer[0] <- k
                    lokasiPlayer[1] <- l
        -> lokasiPlayer

    function jumlahTreasure(map, baris, kolom):
        count <- 0
        for m <- 0 to baris - 1:
            for n <- 0 to kolom - 1:
                if map[m, n] = 'T':
                    count++
        -> count

    function lokasiTreasure(map, baris, kolom, jumlah):
        lokasi <- [jumlah][2]
        count <- 0
        for m <- 0 to baris - 1:
            for n <- 0 to kolom - 1:
                if map[m, n] = 'T':
                    lokasi[count, 0] <- m
                    lokasi[count, 1] <- n
                    count++
        -> lokasi

    function tambahTitik(array, x, y):
        panjang <- (array.Length / 2)
        arrayBaru <- [panjang + 1][2]
        arrayBaru[0, 0] <- x
        arrayBaru[0, 1] <- y
        for i <- 0 to panjang - 1:
```

```
            arrayBaru[i + 1, 0] <- array[i, 0]
            arrayBaru[i + 1, 1] <- array[i, 1]
        -> arrayBaru

    function pernahDikunjungi(array, x, y):
        sudahAda <- false
        for i <- 0 to (array.Length / 2) - 1:
            if array[i, 0] = x AND array[i, 1] = y:
                sudahAda <- true
                break
        -> sudahAda

    FUNCTION cariJalan(map, baris, kolom, player)
        queue <- new Queue()
        tiwal <- new cariBFS()
        titikAwal <- tiwal.cariPlayer(map, baris, kolom)
        jumlahT <- tiwal.jumlahTreasure(map, baris, kolom)
        kujungan <- {{titikAwal[0], titikAwal[1]}}

        queue.Enqueue({{titikAwal[0], titikAwal[1]}})
        ketemu <- FALSE
        pencarian <- TRUE
        nodes <- 0

        WHILE pencarian = TRUE
            pojok <- FALSE
            simpul <- queue.Dequeue()
            xSaatIni <- simpul[0, 0]
            ySaatIni <- simpul[0, 1]

            IF xSaatIni = 0 AND ySaatIni = 0 then
                pojok <- TRUE
                IF map[xSaatIni, ySaatIni + 1] = 'R' OR map[xSaatIni, ySaatIni + 1]
= 'T' OR map[xSaatIni, ySaatIni + 1] = 'K' and ketemu = FALSE AND
tiwal.pernahDikunjungi(kujungan, xSaatIni, (ySaatIni + 1)) = FALSE
                    IF map[xSaatIni, ySaatIni + 1] = 'T'
                        ketemu <- TRUE
                        jumlahT = jumlahT -1
                    cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, (ySaatIni +
1))
                    kujungan <- cariBFS.tambahTitik(kujungan, xSaatIni, (ySaatIni +
1))
                    queue.Enqueue(titikBaru)
                    nodes++

                IF map[xSaatIni + 1, ySaatIni] = 'R' OR map[xSaatIni + 1, ySaatIni]
= 'T' OR map[xSaatIni + 1, ySaatIni] = 'K'and ketemu = FALSE AND
tiwal.pernahDikunjungi(kujungan, (xSaatIni + 1), ySaatIni) = FALSE
                    IF map[xSaatIni + 1, ySaatIni] = 'T'
                        ketemu <- TRUE
                        jumlahT = jumlahT -1
                    cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, (xSaatIni + 1),
ySaatIni)
                    kujungan <- cariBFS.tambahTitik(kujungan, (xSaatIni + 1),
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes++

            else if (xSaatIni = 0 and ySaatIni = kolom - 1) then
                pojok <- true
                if (map[xSaatIni + 1, ySaatIni] = 'R' or map[xSaatIni + 1,
ySaatIni] = 'T' or map[xSaatIni + 1, ySaatIni] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, (xSaatIni + 1), ySaatIni) = false then
                    if (map[xSaatIni + 1, ySaatIni] = 'T') then
                        ketemu <- true
                        jumlahT = jumlahT -1
```

```
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, (xSaatIni + 1),
ySaatIni)
                    kujungan <- cariBFS.tambahTitik(kujungan, (xSaatIni + 1),
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes++
                end if
                if (map[xSaatIni, ySaatIni - 1] = 'R' or map[xSaatIni, ySaatIni -
1] = 'T' or map[xSaatIni, ySaatIni - 1] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, (ySaatIni - 1)) = false then
                    if (map[xSaatIni, ySaatIni - 1] = 'T') then
                        ketemu <- true
                        jumlahT = jumlahT -1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, (ySaatIni -
1))
                    kujungan <- cariBFS.tambahTitik(kujungan, xSaatIni, (ySaatIni -
1))
                    queue.Enqueue(titikBaru)
                    nodes++
                end if
            else if (xSaatIni = baris - 1 and ySaatIni = 0) then
                pojok <- true
                if (map[xSaatIni, ySaatIni + 1] = 'R' or map[xSaatIni, ySaatIni +
1] = 'T' or map[xSaatIni, ySaatIni + 1] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, (ySaatIni + 1)) = false then
                    if (map[xSaatIni, ySaatIni + 1] = 'T') then
                        ketemu <- true
                        jumlahT = jumlahT -1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, (ySaatIni +
1))
                    kujungan <- cariBFS.tambahTitik(kujungan, xSaatIni, (ySaatIni +
1))
                    queue.Enqueue(titikBaru)
                    nodes++
                end if

            if (xSaatIni = baris - 1 and ySaatIni = kolom - 1) then
                pojok <- true
                if (map[xSaatIni, ySaatIni - 1] <- 'R' or map[xSaatIni, ySaatIni -
1] <- 'T' or map[xSaatIni, ySaatIni - 1] <- 'K') and ketemu <- false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, ySaatIni - 1) <- false then
                    if map[xSaatIni, ySaatIni - 1] <- 'T' then
                    ketemu <- true
                    jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, ySaatIni -
1)
                    kujungan <- cariBFS.tambahTitik(kujungan, xSaatIni, ySaatIni -
1)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if
                if (map[xSaatIni - 1, ySaatIni] <- 'R' or map[xSaatIni - 1,
ySaatIni] <- 'T' or map[xSaatIni - 1, ySaatIni] <- 'K') and ketemu <- false and
tiwal.pernahDikunjungi(kujungan, xSaatIni - 1, ySaatIni) <- false then
                    if map[xSaatIni - 1, ySaatIni] <- 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni - 1,
```

```
ySaatIni)
                    kujungan <- cariBFS.tambahTitik(kujungan, xSaatIni - 1,
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if
            else if (xSaatIni = 0 and pojok = false) then
                if (map[xSaatIni, ySaatIni + 1] = 'R' or map[xSaatIni, ySaatIni +
1] = 'T' or map[xSaatIni, ySaatIni + 1] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, ySaatIni + 1) = false then
                    if map[xSaatIni, ySaatIni + 1] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS = new cariBFS()
                    titikBaru = cariBFS.tambahTitik(simpul, xSaatIni, ySaatIni + 1)
                    kujungan <- cariBFS.tambahTitik(kujungan, xSaatIni, ySaatIni +
1)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni + 1, ySaatIni] = 'R' or map[xSaatIni + 1,
ySaatIni] = 'T' or map[xSaatIni + 1, ySaatIni] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni + 1, ySaatIni) = false then
                    if map[xSaatIni + 1, ySaatIni] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni + 1,
ySaatIni)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni + 1,
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni, ySaatIni - 1] = 'R' or map[xSaatIni, ySaatIni -
1] = 'T' or map[xSaatIni, ySaatIni - 1] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, ySaatIni - 1) = false then
                    if map[xSaatIni, ySaatIni - 1] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, ySaatIni -
1)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni, ySaatIni -
1)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

            else if (xSaatIni = baris - 1 and pojok = false) then
                if (map[xSaatIni, ySaatIni + 1] = 'R' or map[xSaatIni, ySaatIni +
1] = 'T' or map[xSaatIni, ySaatIni + 1] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, ySaatIni + 1) = false then
                    if map[xSaatIni, ySaatIni + 1] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, ySaatIni +
1)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni, ySaatIni +
1)
                    queue.Enqueue(titikBaru)
```

```
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni, ySaatIni - 1] = 'R' or map[xSaatIni, ySaatIni -
1] = 'T' or map[xSaatIni, ySaatIni - 1] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, ySaatIni - 1) = false then
                    if map[xSaatIni, ySaatIni - 1] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, ySaatIni -
1)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni, ySaatIni -
1)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni - 1, ySaatIni] = 'R' or map[xSaatIni - 1,
ySaatIni] = 'T' or map[xSaatIni - 1, ySaatIni] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni - 1, ySaatIni) = false then
                    if map[xSaatIni - 1, ySaatIni] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni - 1,
ySaatIni)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni - 1,
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

            else if (ySaatIni = kolom - 1 and pojok = false) then
                if (map[xSaatIni + 1, ySaatIni] = 'R' or map[xSaatIni + 1,
ySaatIni] = 'T' or map[xSaatIni + 1, ySaatIni] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni + 1, ySaatIni) = false then
                    if map[xSaatIni + 1, ySaatIni] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni + 1,
ySaatIni)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni + 1,
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni, ySaatIni - 1] = 'R' or map[xSaatIni, ySaatIni -
1] = 'T' or map[xSaatIni, ySaatIni - 1] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, ySaatIni - 1) = false then
                    if map[xSaatIni, ySaatIni - 1] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, ySaatIni -
1)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni, ySaatIni -
1)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if
```

```
                if (map[xSaatIni - 1, ySaatIni] = 'R' or map[xSaatIni - 1,
ySaatIni] = 'T' or map[xSaatIni - 1, ySaatIni] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni - 1, ySaatIni) = false then
                    if map[xSaatIni - 1, ySaatIni] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni - 1,
ySaatIni)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni - 1,
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

            else if (ySaatIni = 0 and pojok = false) then
                if (map[xSaatIni, ySaatIni + 1] = 'R' or map[xSaatIni, ySaatIni +
1] = 'T' or map[xSaatIni, ySaatIni + 1] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, ySaatIni + 1) = false then
                    if map[xSaatIni, ySaatIni + 1] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, ySaatIni +
1)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni, ySaatIni +
1)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni + 1, ySaatIni] = 'R' or map[xSaatIni + 1,
ySaatIni] = 'T' or map[xSaatIni + 1, ySaatIni] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni + 1, ySaatIni) = false then
                    if map[xSaatIni + 1, ySaatIni] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni + 1,
ySaatIni)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni + 1,
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni - 1, ySaatIni] = 'R' or map[xSaatIni - 1,
ySaatIni] = 'T' or map[xSaatIni - 1, ySaatIni] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni - 1, ySaatIni) = false then
                    if map[xSaatIni - 1, ySaatIni] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni - 1,
ySaatIni)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni - 1,
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

            else
                if (map[xSaatIni, ySaatIni + 1] = 'R' or map[xSaatIni, ySaatIni +
1] = 'T' or map[xSaatIni, ySaatIni + 1] = 'K') and ketemu = false and
```

```
tiwal.pernahDikunjungi(kujungan, xSaatIni, ySaatIni + 1) = false then
                    if map[xSaatIni, ySaatIni + 1] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, ySaatIni +
1)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni, ySaatIni +
1)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni + 1, ySaatIni] = 'R' or map[xSaatIni + 1,
ySaatIni] = 'T' or map[xSaatIni + 1, ySaatIni] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni + 1, ySaatIni) = false then
                    if map[xSaatIni + 1, ySaatIni] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni + 1,
ySaatIni)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni + 1,
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni, ySaatIni - 1] = 'R' or map[xSaatIni, ySaatIni -
1] = 'T' or map[xSaatIni, ySaatIni - 1] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni, ySaatIni - 1) = false then
                    if map[xSaatIni, ySaatIni - 1] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni, ySaatIni -
1)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni, ySaatIni -
1)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if

                if (map[xSaatIni - 1, ySaatIni] = 'R' or map[xSaat ini - 1,
ySaatIni] = 'T' or map[xSaatIni - 1, ySaatIni] = 'K') and ketemu = false and
tiwal.pernahDikunjungi(kujungan, xSaatIni - 1, ySaatIni) = false then
                    if map[xSaatIni - 1, ySaatIni] = 'T' then
                        ketemu <- true
                        jumlahT <- jumlahT - 1
                    end if
                    cariBFS cariBFS <- new cariBFS()
                    titikBaru <- cariBFS.tambahTitik(simpul, xSaatIni - 1,
ySaatIni)
                    kujungan = cariBFS.tambahTitik(kujungan, xSaatIni - 1,
ySaatIni)
                    queue.Enqueue(titikBaru)
                    nodes <- nodes + 1
                end if
        if ketemu = true then
            harusDequeue <- queue.count
            for i <- 1 to harusDequeue
                queue.Dequeue()
            map[queue.ElementAt(0)[0,0],queue.ElementAt(0)[0,1]] <- 'R'

            if jumlahT > 0 then
```

```
                ketemu <- false
                kunjungan <- [,]
                kunjungan[0,0] <- queue.ElementAt(0)[0,0]
                kunjungan[0,1] <- queue.ElementAt(0)[0,1]
            else
                pencarian <- FALSE
            end if
        end if

        hasil <- queue.duqueue()
        tambah <- new cariBFS()
        hasil <- tambah.tambahTitik(hasil, hasil[0,0], hasil[0,1])

        for each i in range from 0 to (hasil.Length/2)/2
            temp <- hasil[i, 0]
            hasil[i, 0] <- hasil[hasil.Length/2 - i - 1, 0]
            hasil[hasil.Length/2 - i - 1, 0] <- temp

            temp1 <- hasil[i, 1]
            hasil[i, 1] <- hasil[hasil.Length/2 - i - 1, 1]
            hasil[hasil.Length/2 - i - 1, 1] <- temp1
        end for

        -> hasil
    end function
```

2) Pseudocode untuk algoritma DFS

```
class Point
        x: integer
        y: integer

        constructor Point()
             x <- 0
             y <- 0

        constructor Point(x: integer, y: integer)
             this.x <- x
             this.y <- y

        function Equals(obj: object) returns boolean
             if obj is null or obj.GetType() is not Point
                  return false

             temp <- obj as Point
                   -> x = temp.x and y = temp.y

        function GetHashCode() returns integer
             -> (x + y).GetHashCode()

class Algorithm
        function stackTreasure(arr: array of char) returns Stack of Point
             treasure <- new Stack<Point>
             for each index row of array
                  for each index column of array
                      if array(index)is equal to 'T' then
                          treasure.Push(new Point(i, j))
             -> treasure

        function printStep(s: Stack of Point, count: integer) returns string
             direction <- ""
             if s.Count = 1 then
                 return ""
             temp <- s.Peek()
```

```
            s.Pop()
            temp1 <- s.Peek()
            direction <- printStep(s, count) + checkAdjacent(temp1, temp)
            s.Push(temp)
            -> direction

        procedure printStack(s: Stack of Point)
            if s is empty then
                return
            temp <- s.Peek()
            s.Pop()
            printStack(s)
            Output(temp.x + "," + temp.y)
            s.Push(temp)

        procedure printArray(a : array of Point)
            for each index of array
                Output("(" + a[i].x + "," + a[i].y + ")")

        procedure treasureArrived(s: Stack of Point, p: Point)
            if s is empty then
                return
            temp <- s.Peek()
            s.Pop()
            treasureArrived(s, p)
            if p.x != temp.x OR p.y != temp.y then
                s.Push(temp)

        procedure printMatrix(arr: array of char)
            for each index row of array
                for each index column of array
                    Output(arr[i, j] + " ")

        function findK(arr: array of char) returns Point
            p <- new Point()
            for each index row of array
                for each index column of array
                    if array(index)is equal to 'K' then
                        p <- new Point(i, j)
            -> p

        procedure  adjacentPoint(arr:  array  of  char,  visited:  Stack  of  Point,
        unvisited: Stack of Point)
            count <- 0
            p <- visited.Peek()
            if p.x != 0 AND arr[p.x - 1, p.y] != 'X' AND NOT visited.Contains(new
    Point(p.x - 1, p.y)) then
                p1 <- new Point(p.x - 1, p.y)
                unvisited.Push(p1)
                count <- count + 1
            if p.y != 0 AND arr[p.x, p.y - 1] != 'X' AND NOT visited.Contains(new
    Point(p.x, p.y - 1)) then
                p1 <- new Point(p.x, p.y - 1)
                unvisited.Push(p1)
                count <- count + 1
            if p.x != arr.GetLength(0) - 1 AND arr[p.x + 1, p.y] != 'X' AND NOT
    visited.Contains(new Point(p.x + 1, p.y)) then
                p1 <- new Point(p.x + 1, p.y)
                unvisited.Push(p1)
                count <- count + 1
            if p.y != arr.GetLength(1) - 1 AND arr[p.x, p.y + 1] != 'X' AND NOT
    visited.Contains(new Point(p.x, p.y + 1)) then
                p1 <- new Point(p.x, p.y + 1)
                unvisited.Push(p1)
                count <- count + 1
            if count = 0 then
                Output("BackTracking")
```

```
procedure move(visited: Stack of Point, unvisited: Stack of Point, path: Stack
of Point)
     temp <- unvisited.Pop()
     visitedCopy <- create a copy of visited stack
     Reverse(visitedCopy)
     if not checkAdjacentBool(temp, visited.Peek()) then
         Output("masuk sini")
         temp1 <- visitedCopy.Pop()
         temp2 <- visitedCopy.Pop()
         temp4 <- visited.Pop()
         path.Push(temp2)
         Output("Temp: " + temp.x + "," + temp.y)
         while not checkAdjacentBool(temp, temp2) do
             temp2 <- visitedCopy.Pop()
             temp4 <- visited.Pop()
             path.Push(temp2)
     visited.Push(temp)
     path.Push(temp)

function checkAdjacent(p1: Point, p2: Point) returns string
     if (p1.x = p2.x + 1 and p1.y = p2.y) then
         direction <- "U"
     else if (p1.x = p2.x - 1 and p1.y = p2.y) then
         direction <- "D"
     else if (p1.x = p2.x and p1.y = p2.y + 1) then
         direction <- "L"
     else if (p1.x = p2.x and p1.y = p2.y - 1) then
         direction <- "R"
     else
         direction <- "Backtracking sampai di (" + p2.x + "," + p2.y + ")"
     -> direction

function checkAdjacentBool(p1: Point, p2: Point) returns boolean
     if (p1.x = p2.x + 1 and p1.y = p2.y) then
         flag <- true
     else if (p1.x = p2.x - 1 and p1.y = p2.y) then
         flag <- true
     else if (p1.x = p2.x and p1.y = p2.y + 1) then
         flag <- true
     else if (p1.x = p2.x and p1.y = p2.y - 1) then
         flag <- true
     else:
         flag <- false
     -> flag

function countNode(s: Stack of Point) returns integer
     if s is empty then
         return 0
     temp <- s.Peek()
     s.Pop()
     countNode(s)
     if not s.Contains(temp):
         s.Push(temp)
     -> s.Count

procedure Reverse(s: Stack of Point)
     if s is empty then
         return
     element <- s.Pop()
     Reverse(s)
     InsertAndRearrange(s, element)

procedure InsertAndRearrange(s: Stack of Point, element: Point)
     if s is empty then
         s.Push(element)
     else
         temp <- s.Pop()
         InsertAndRearrange(s, element)
```

```
                s.Push(temp)
```

II. Struktur Data dan Spesifikasi Program

Struktur data yang dipergunakan dalam program kami adalah sebagai berikut:

1) Graph

Penggunaan graph yang dalam program kami diimplementasikan pada bagian pencarian seluruh *treasure* dengan segala kemungkinan jalan yang ada pada *maze*. Untuk *nodes* dalam graph berupa semua kemungkinan *grid* pada *maze* yang dapat dilewati untuk mencari *treasure* dan *edges* berupa proses perpindahan dari satu *grid* pada *maze* ke *grid* lainnya hingga semua harta karun telah ditemukan.

2) Queue

Penggunaan *queue* dalam program kami diimplementasikan pada bagian pencarian *treasure* dengan algoritma BFS. Implementasi digunakan untuk menyimpan antrian yaitu berupa *node* (kotak *grid*) yang telah dikunjungi. Penyimpanan dilakukan dengan aturan *First In First Out (FIFO)* dengan memasukkan *grid* yang dikunjungi dari belakang antrian *Queue* (*Enqueue*) dan mengeluarkan *grid* yang akan dicek dari depan antrian (*Dequeue*).

3) Stack

Penggunaan *stack* dalam program kami diimplementasikan pada bagian pencarian *treasure* dengan algoritma DFS. Implementasi digunakan untuk menyimpan antrian yaitu berupa *node* (kotak *grid*) yang telah dikunjungi. Penyimpanan dilakukan dengan aturan *Last In First Out (LIFO)* dengan memasukkan *grid* yang dikunjungi dari atas tumpukan *Stack* (*Push*) dan mengeluarkan *grid* yang akan dicek dari atas tumpukan juga (*Pop*).

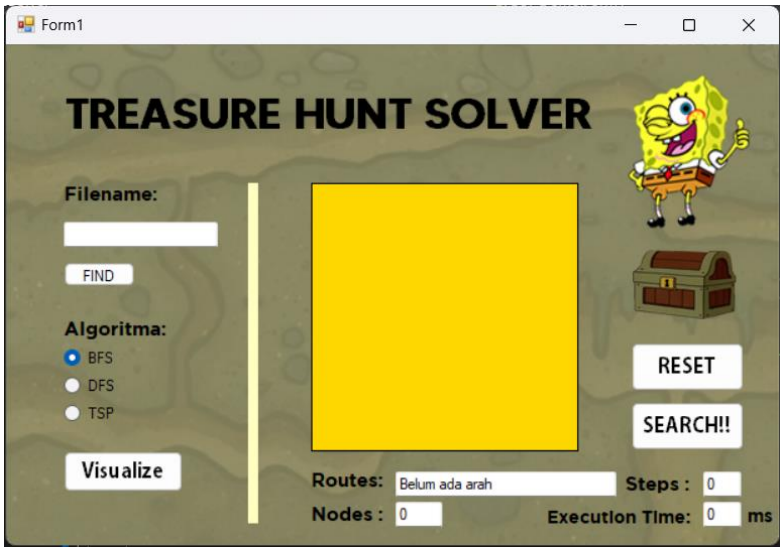
III. Tata Cara Penggunaan Program

Untuk dapat menjalankan program, diperlukan beberapa spesifikasi sebagai berikut:

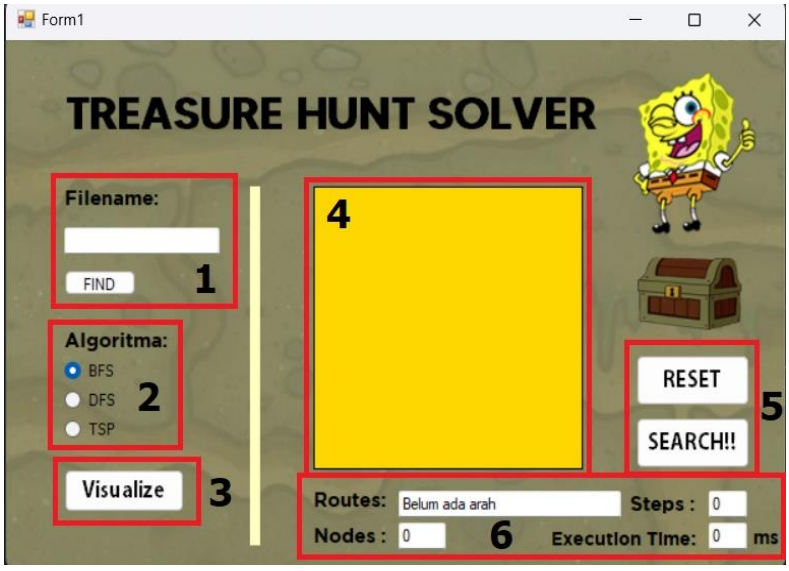
1) Visual Studio 2022
2) .NET 7.0
3) Menjalankan program dengan Windows Operating System

Setelah semua spesifikasi terpenuhi, langkah-langkah untuk dapat menjalankan program adalah sebagai berikut:

1) Clone repository program terlebih dahulu, link github dicantumkan pada akhir laporan
2) Buka file "../Tubes2_Xixi_src_MazeSolver.sln
3) Jalankan program dengan mengklik tombol F5 atau langsung run pada bagian atas aplikasi Visual Studio
4) Setelah program dijalankan, akan muncul tampilan sebagai berikut:



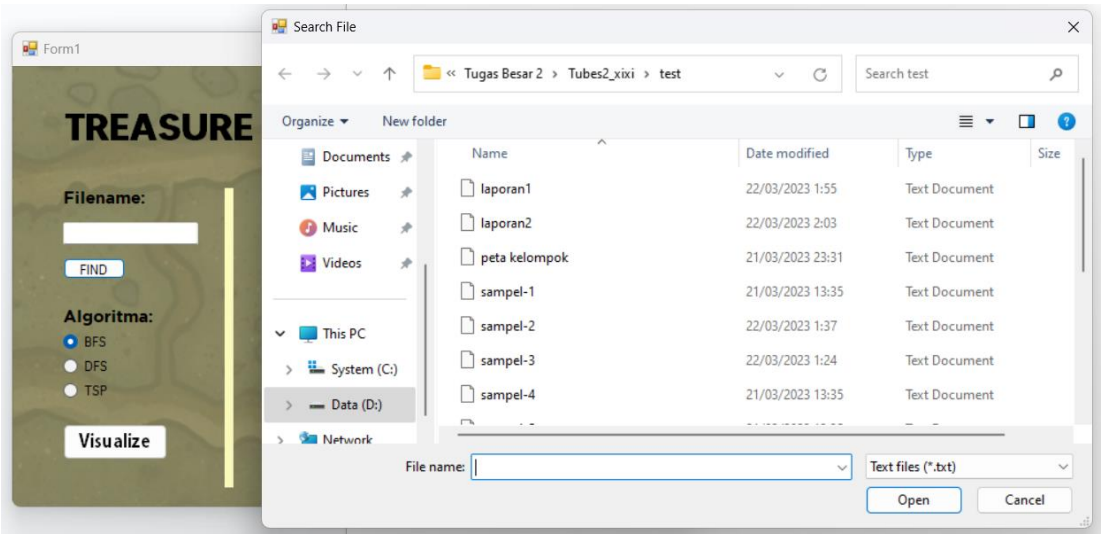
5) Keterangan yang ada pada *tools* program



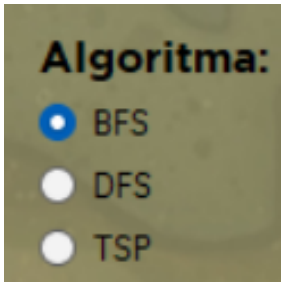
Keterangan:

1. *Button* find untuk menginput file.txt yang akan dicari bentuk *maze*-nya
2. *Toggle* untuk memilih proses menjalankan algoritma dengan BFS, DFS, atau TSP
3. *Button* untuk menampilkan visualisasi *maze*
4. *Grid* untuk memproyeksikan *maze* dari *file*
5. *Button reset* untuk mengembalikan *grid* ke dalam bentuk *default* kosong dan *search* untuk melakukan pencarian dengan metode yang telah dipilih pada *toggle* sebelumnya
6. *Textbox* untuk menampilkan urutan rute, jumlah nodes yang diperiksa, langkah pencarian, dan waktu eksekusi program

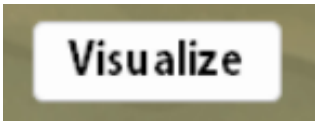
6) Melakukan pencarian file.txt dengan meng-klik tombol *find*



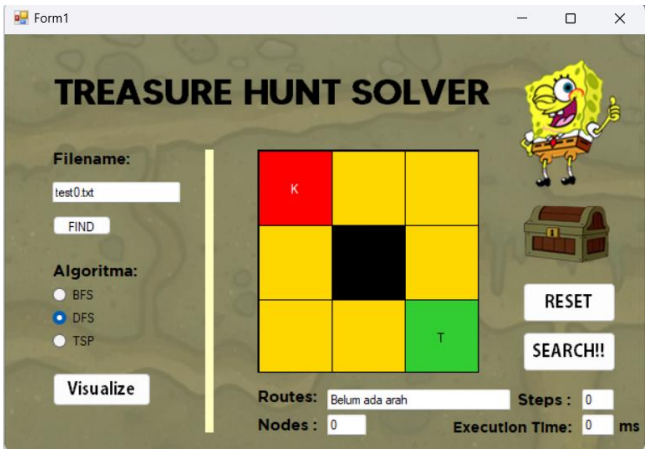
7) Menentukan algoritma yang dipakai antara BFS dan DFS



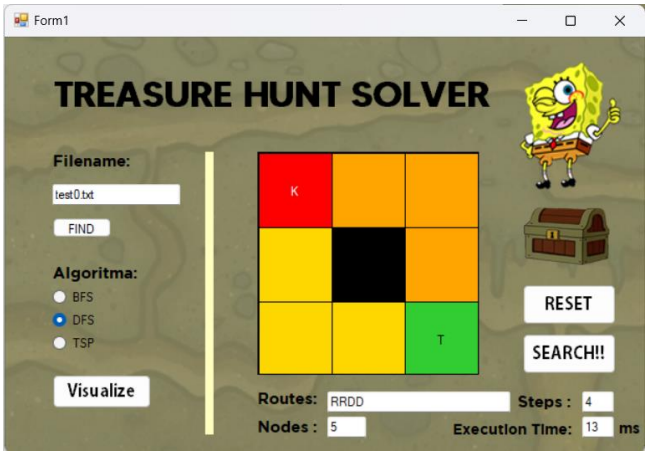
8) Mengklik tombol visualisasi untuk menampilkan bentuk *maze*



9) Bentuk *maze* akan ditampilkan pada grid



10) Melakukan pencarian rute terdekat dengan mengklik *search*

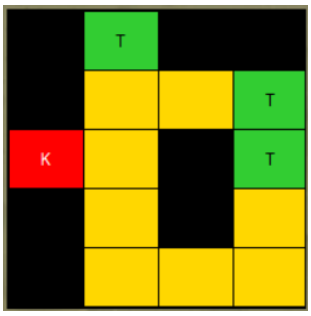


11) Mengklik tombol *reset* untuk mengembalikan *grid* ke bentuk *default*

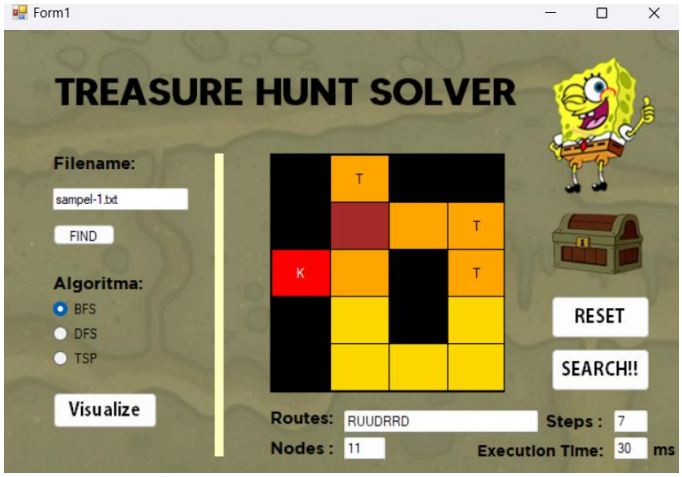
IV. Hasil Pengujian

1) Pengujian untuk test case pertama (sample-1)

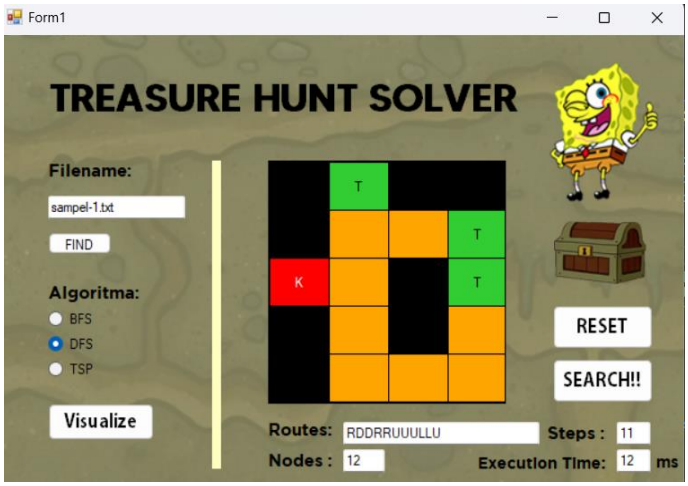
```
X T X X
X R R T
K R X T
X R X R
X R R R
```



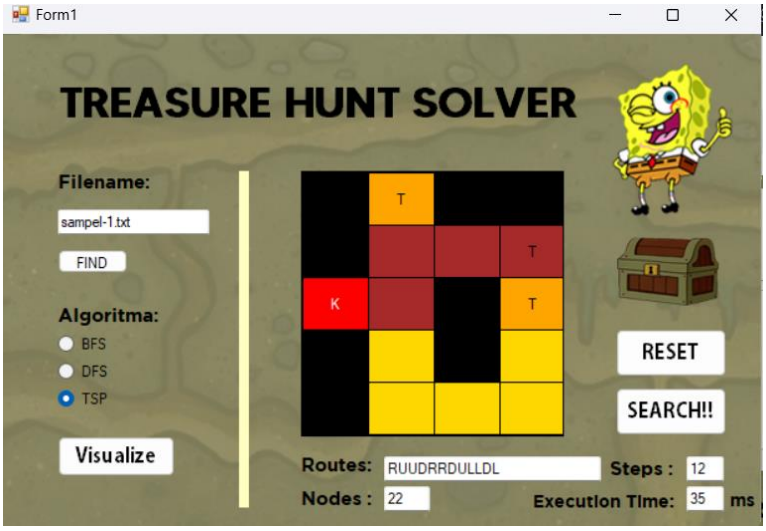
a) Algoritma BFS



b) Algoritma DFS

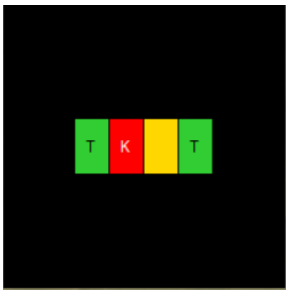


c) Algoritma TSP

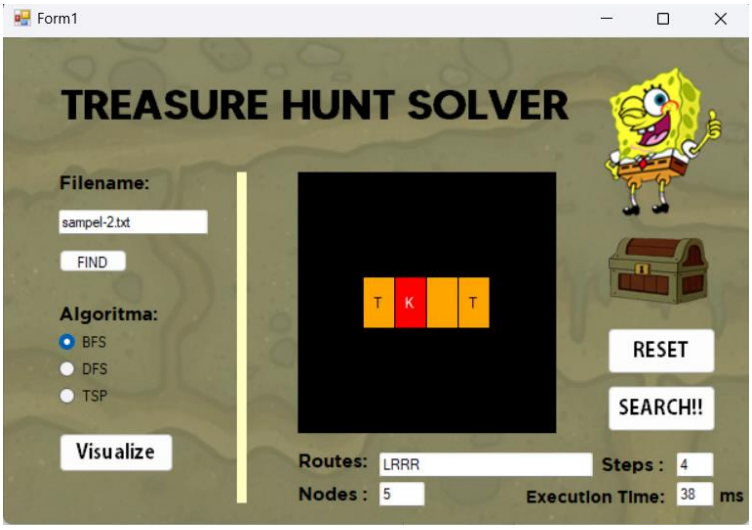


2) Pengujian untuk test case kedua (sample-2)

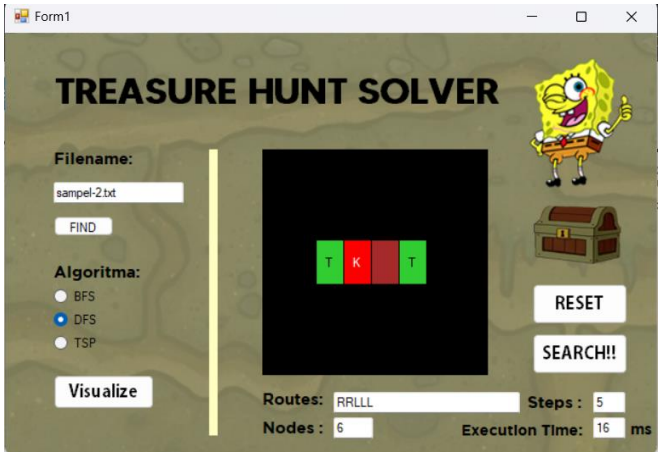
```
X X X X X X X X
X X X X X X X X
X X T  K R T X X
X X X X X X X X
X X X X X X X X
```



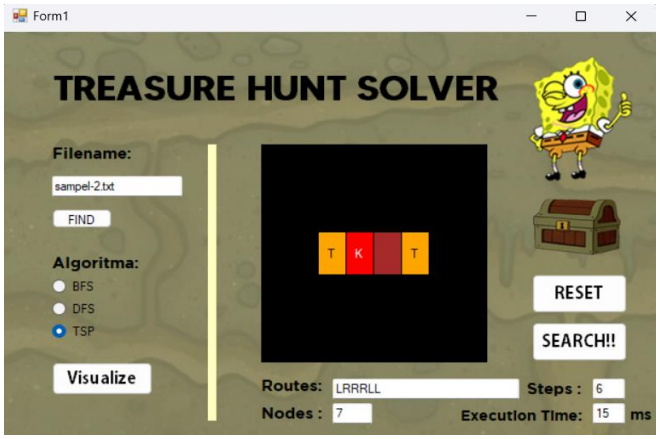
a) Algoritma BFS



b) Algoritma DFS

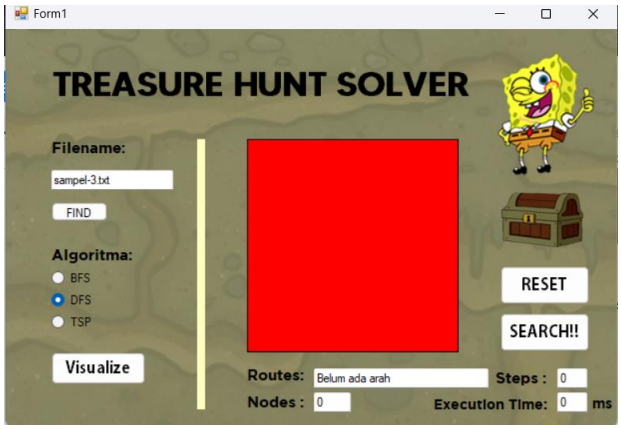


c) Algoritma TSP



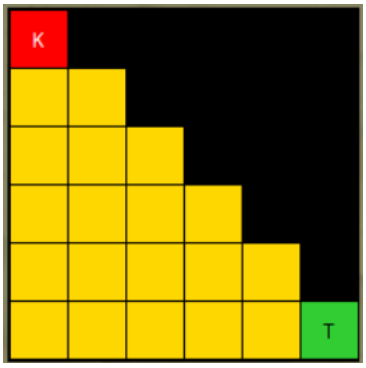
3) Pengujian untuk test case ketiga (sample-3)

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

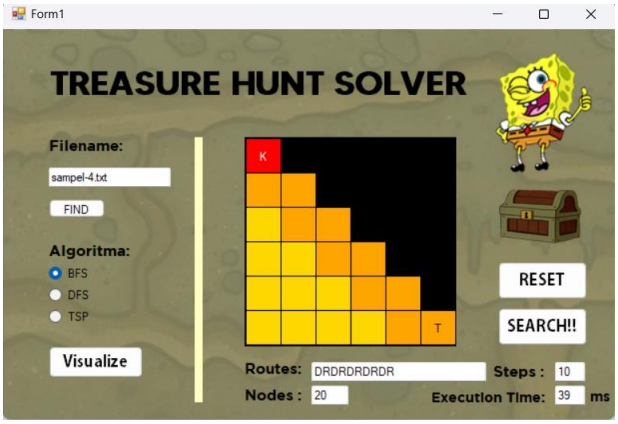


4) Pengujian untuk test case keempat (sample-4)

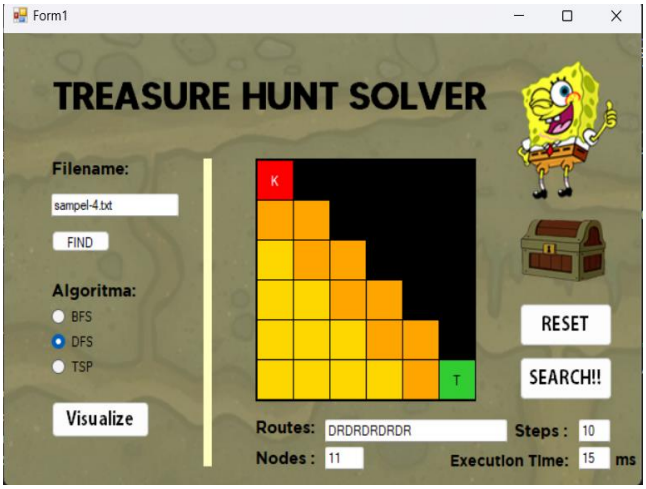
```
K X X X X X
R R X X X X
R R R X X X
R R R R X X
R R R R R X
R R R R R T
```



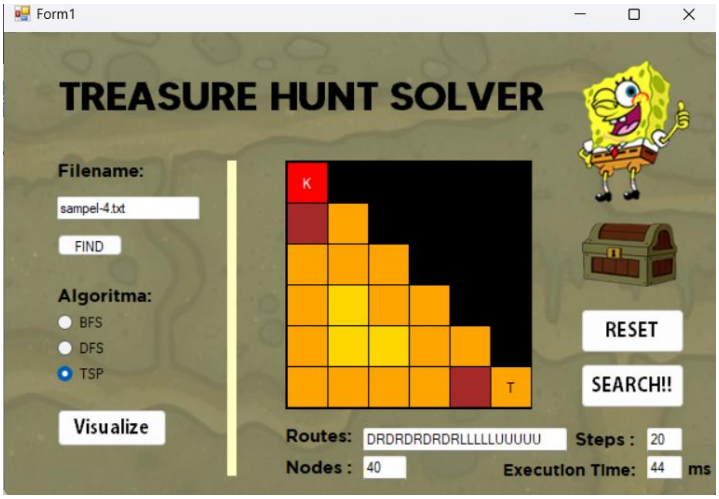
a) Algoritma BFS



b) Algoritma DFS

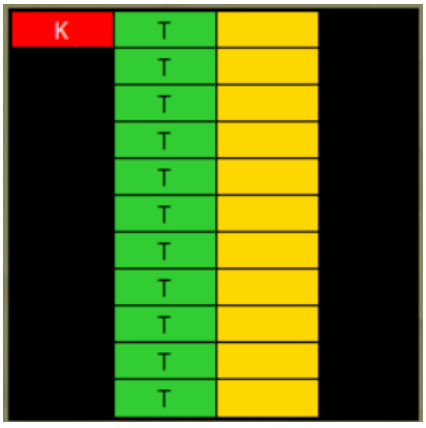


c) Algoritma TSP

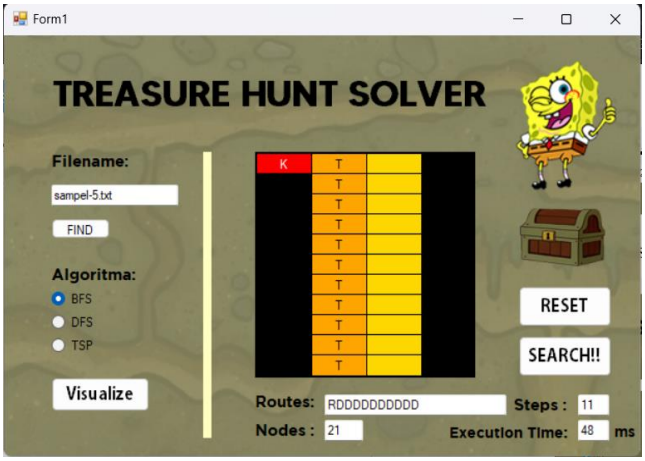


5) Pengujian untuk test case kelima (sample-5)

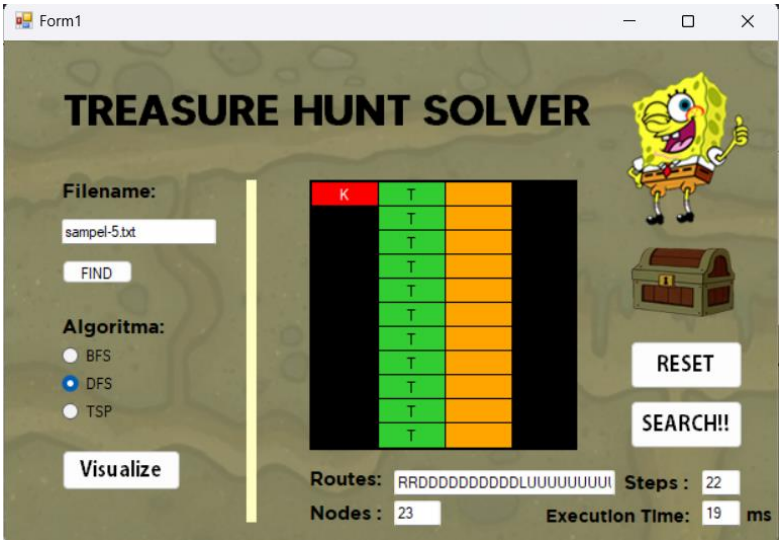
```
K T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
X T R X
```



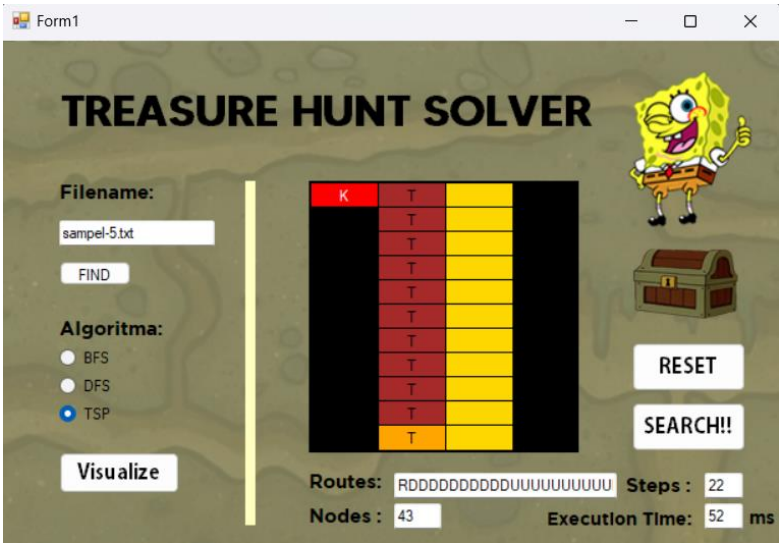
a. Algoritma BFS



b. Algoritma DFS



c. Algoritma TSP



V. Analisis Desain Solusi Algoritma BFS dan DFS

Berdasarkan hasil pengujian 1, algoritma DFS dengan prioritas *Right* (Kanan), *Down* (Bawah), *Left* (Kiri), *Up* (Atas), mencari *treasure* dengan *path* sebagai berikut: Start (2,0) → Right (2,1) → Down (3,1) → Down (4,1) → Right (4,2) → Right (4,3) → Up (3,3) → Up (2,3) → Up (1,3) → Left (1,2) → Left (1,1) → Up (0,1). Nodes yang dilalui berjumlah 12

dengan *execution time* 12 ms. Sedangkan, algoritma BFS dengan prioritas *Right* (Kanan), *Down* (Bawah), *Left* (Kiri), *Up* (Atas), mencari *treasure* dengan *path* sebagai berikut: Start (2,0) → Right (2,1) → Up (1,1) → Up (0,1) → Down (1,1) → Right (1,2) → Right (1,3) → Down (2,3). Nodes yang dilalui berjumlah 11 dengan *execution time* 30 ms. Pada *map* ini, algoritma BFS melakukan *backtracking* untuk mencari nodes dimana *treasure* berada.

Berdasarkan hasil pengujian 2, algoritma DFS dengan prioritas *Right* (Kanan), *Down* (Bawah), *Left* (Kiri), *Up* (Atas), mencari treasure dengan *path* sebagai berikut: Start (2,3) → Right (2,4) → Right (2,5) → Left (2,4) → Left (2,3) → Left (2,2). Nodes yang dilalui berjumlah 6 dengan *execution time* 16 ms. Pada *map* ini, algoritma DFS melakukan *backtracking* untuk mencari nodes dimana *treasure* berada. Sedangkan, algoritma BFS dengan prioritas *Right* (Kanan), *Down* (Bawah), *Left* (Kiri), *Up* (Atas), mencari treasure dengan *path* sebagai berikut: Start (2,3) → Left (2,2) → Right (2,3) → Right (2,4) → Right (2,5). Nodes yang dilalui berjumlah 5 dengan *execution time* 38 ms. Pada *map* ini, algoritma BFS juga melakukan *backtracking*.

Dari hasil tersebut, dapat disimpulkan bahwa pencarian dengan algoritma DFS memiliki keunggulan dalam efektivitas waktu dan ruang dikarenakan algoritma BFS memiliki kompleksitas sebesar O(|E| + |V|), sedangkan algoritma DFS hanya memiliki kompleksitas sebesar O(V) sehingga lebih mangkus. Sebaliknya, algoritma BFS mampu mencari *treasure* dengan *path* yang lebih sedikit jika dibandingkan dengan algoritma DFS.

BAB V

KESIMPULAN DAN SARAN

I. Kesimpulan

Dari tugas besar ini dapat disimpulkan hal-hal sebagai berikut,

1) Algoritma BFS dan DFS dapat digunakan untuk proses pencarian rute untuk mencapai *goal* akhir tertentu, seperti pada contoh tugas besar ini, yaitu untuk menemukan semua harta karun.
2) Proses pencarian dengan algoritma BFS memakan waktu yang lebih lama jika dibandingkan dengan algoritma DFS

II. Saran

Saran untuk penyelesaian masalah ini untuk kedepannya adalah sebagai berikut,

1) Terdapat beberapa kondisi tertentu yang sebaiknya dispesifikasikan lebih jelas, seperti kondisi ketika tidak dapat mencapai *treasure* ataupun kondisi pencarian *backtracking* untuk algoritma BFS
2) Terdapat beberapa kondisi tertentu yang masih belum terlalu optimal secara kompleksitas program sehingga program dapat dikembangkan kedepannya agar menjadi lebih baik lagi.

III. Refleksi

Dari pengerjaan tugas besar ini, hal penting yang didapat adalah mengenai pentingnya komunikasi antar anggota kelompok, mengingat program dibuat secara bersama-sama dengan bagian masing-masing. Seperti pada proses penggabungan *front-end* dengan *back-end*, diperlukan komunikasi yang baik agar program dapat menyatu dengan baik. Selain itu, dalam tugas besar ini, juga memberikan pelajaran mengenai algoritma BFS dan DFS yang tentunya memiliki banyak aplikasi pada berbagai macam program yang terkait dengan proses pencarian di masa yang akan datang.

IV. Tanggapan

Tugas yang diberikan sudah sangat membantu mahasiswa dalam mengerti terkait algoritma BFS dan DFS. Tugasnya juga cukup unik, menarik, dan inovatif. Tetap dipertahankan dan bahkan ditingkatkan untuk pemberian tugas-tugas kedepannya agar dapat semakin lebih baik lagi.

DAFTAR PUSTAKA

[1] R. Munir (2021). Breadth/Depth First Search (BFS/DFS) Bagian 1 [Powerpoint Slides]. Available:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf

[2] R. Munir (2021). Breadth/Depth First Search (BFS/DFS) Bagian 2 [Powerpoint Slides]. Available:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf

[3] M. Napizahni. "Apa Itu .NET Framework? Pengertian, Sejarah, dan Fungsinya." Dewaweb. https://www.dewaweb.com/blog/apa-itu-net-framework/ (Diakses pada tanggal 21 Maret 2023)

[4] "Tentang Microsoft Visual Studio dan Kegunaannya." Proxsis Surabaya. https://surabaya.proxsisgroup.com/tentang-microsoft-visual-studio-dan-kegunaannya/ (Diakses pada tanggal 21 Maret 2023)

[5] Surya, A. A. "Apakah itu Windows Form ?" IDCSharp. https://idcsharp.com/2019/04/27/apakah-itu-windows-form-winform/ (Diakses pada tanggal 21 Maret 2023)

IF2211
Strategi Algoritma

LAMPIRAN

Pranala Github : https://github.com/jasonrivalino/Tubes2_xixi

Video Demonstrasi : https://youtu.be/22WJvKkyisw