

LAPORAN TUGAS BESAR II

Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt

Laporan dibuat untuk memenuhi salah satu tugas mata kuliah

IF2211 Strategi Algoritma

[FOTO]

Disusun oleh :

uburubur

Angger Ilham Amanullah 13521001

Kelvin Rayhan Alkarim 13521005

Maggie Zeta RS 13521117

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

Bab 1.....	3
Bab 2.....	7
2.1 Graph Traversal.....	7
2.2 Breadth-First Search (BFS).....	7
2.3 Depth-First Search (DFS).....	7
2.4 C# Desktop Application Development.....	7
Bab 3.....	9
3.1 Langkah-langkah Pemecahan Masalah.....	9
3.2 Proses Mapping Persoalan.....	9
3.2.1 Proses Mapping Persoalan dengan Algoritma BFS.....	9
3.2.2 Proses Mapping Persoalan dengan Algoritma DFS.....	10
3.3 Contoh Ilustrasi Kasus Lain.....	10
Ada beberapa kasus maze yang dapat terjadi antara lain :.....	10
Bab 4.....	12
4.1 Implementasi Program Utama.....	12
4.2 Penjelasan Struktur Data.....	32
4.3 Tata Cara Penggunaan Program.....	33
4.4 Hasil Pengujian.....	33
4.5 Analisis Desain Solusi.....	35
Bab 5.....	36
5.1 Kesimpulan.....	36
5.2 Saran.....	36
5.3 Refleksi.....	36
5.4 Tanggapan.....	36
Lampiran.....	37
Daftar Pustaka.....	38

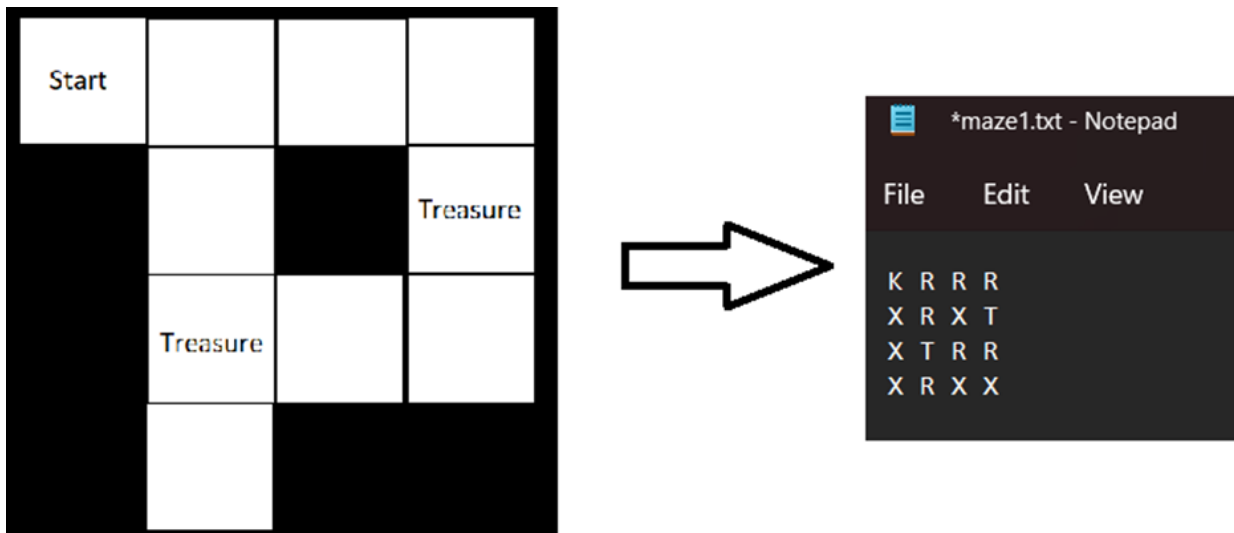
Bab 1

Deskripsi Tugas

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

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

Contoh file input :

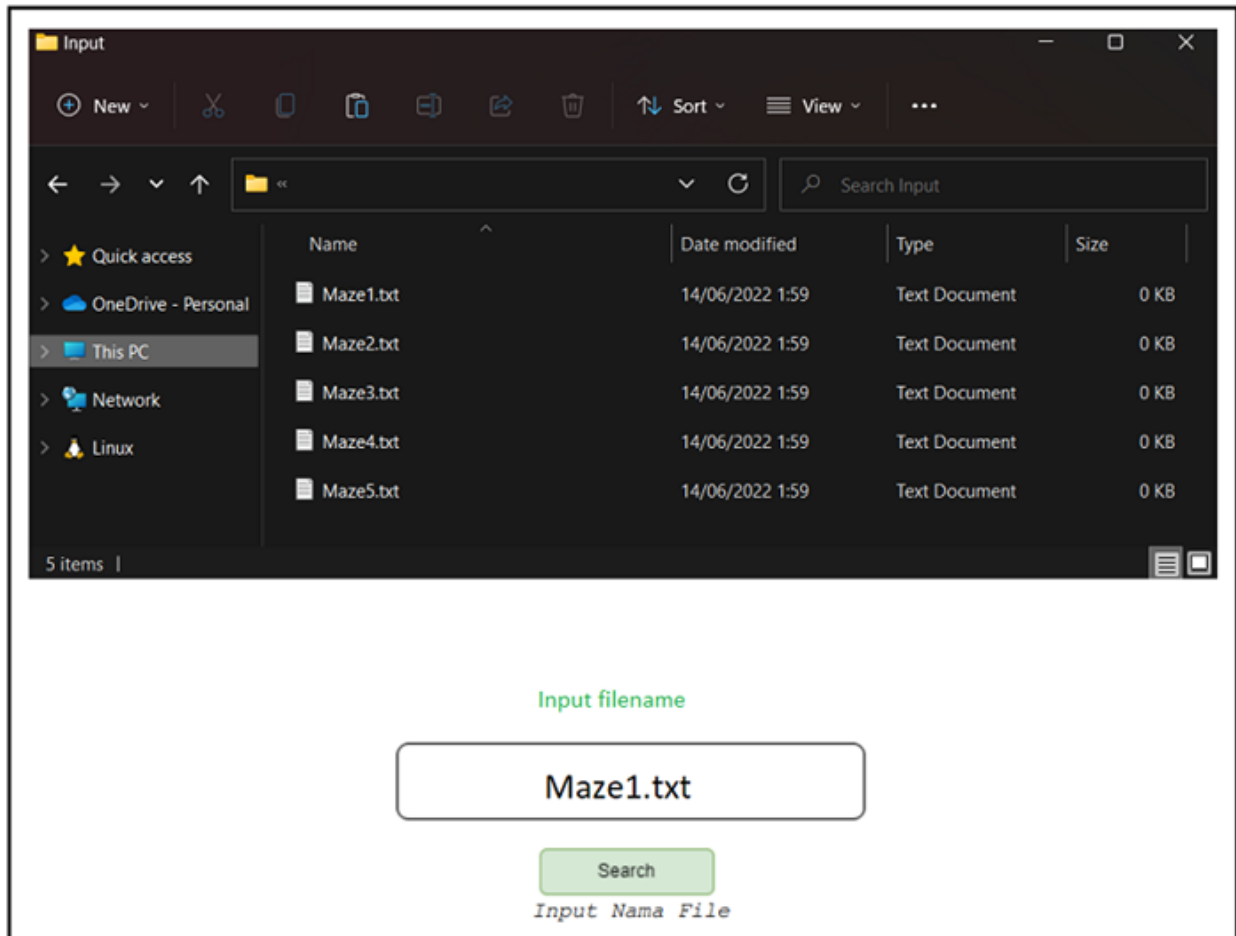


Gambar 2. Ilustrasi input file maze

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 :

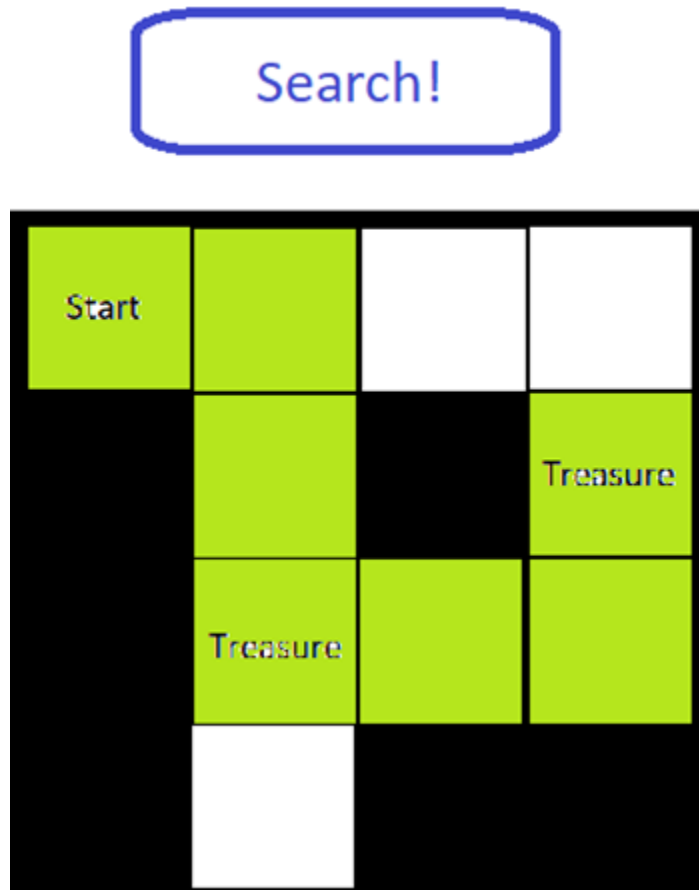


Gambar 3. Contoh input program

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 handle 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 DFS dan BFS. Setelah tombol diklik, program akan melakukan pemberian warna pada rute solusi.

Bab 2

Landasan Teori

2.1 Graph Traversal

Graph traversal adalah suatu proses yang melibatkan kunjungan satu per satu simpul pada suatu graf. Proses ini bisa dilakukan untuk berbagai tujuan, seperti untuk pemeriksaan atau untuk mengubah nilai pada simpul-simpul tersebut. Umumnya, graph traversal dilakukan dengan mengunjungi simpul-simpul yang bertetangga dengan simpul awal yang diperiksa. Ada dua jenis graph traversal yang umum digunakan, yaitu BFS (Breadth-First Search) dan DFS (Depth-First Search), dan kedua jenis ini akan dijelaskan lebih lanjut pada sub-bab berikutnya.

2.2 Breadth-First Search (BFS)

Breadth-First Search (BFS) adalah suatu algoritma graph traversal yang bekerja dengan cara mengunjungi setiap simpul secara melebar. Proses ini dimulai dengan mengunjungi sebuah simpul awal dan kemudian melanjutkan kunjungan ke seluruh simpul yang belum dikunjungi dan juga bertetangga dengan simpul awal tersebut. Dalam implementasinya, BFS dapat dijalankan dengan menggunakan struktur data queue sebagai bantuannya.

2.3 Depth-First Search (DFS)

Depth-First Search (DFS) merupakan salah satu algoritma graph traversal yang melakukan penelusuran berdasarkan kedalaman pohon. Penelusuran dimulai dari root dan dilanjutkan ke salah satu simpul anaknya, dengan prioritas penelusuran berdasarkan anak pertama yaitu yang paling kiri. Kemudian, penelusuran dilanjutkan melalui simpul anak pertama dari simpul anak pertama level sebelumnya hingga mencapai level terdalam.

Setelah mencapai level terdalam, penelusuran akan kembali ke level sebelumnya untuk menelusuri simpul anak kedua pada pohon, dan kemudian dilakukan penelusuran lagi hingga mencapai level terdalam. Proses ini diulang kembali hingga semua simpul telah dikunjungi.

Dalam implementasinya, DFS dapat dilakukan dengan menggunakan cara rekursif atau dengan struktur data stack.

2.4 C# Desktop Application Development

C# adalah bahasa pemrograman yang dibuat oleh Microsoft dan didesain untuk menulis program yang berjalan pada framework .NET. Bahasa pemrograman C# digunakan secara

luas untuk mengembangkan aplikasi Windows dan juga digunakan dalam pengembangan program aplikasi web dan game. Seperti halnya bahasa pemrograman berorientasi objek lainnya, C# juga menerapkan konsep-konsep objek seperti inheritance, class, polymorphism, encapsulation, dan lain-lain.

Dalam penggunaan bahasa C#, developer biasanya menggunakan IDE Visual Studio 2022. Untuk pengembangan aplikasi desktop, Visual Studio menyediakan berbagai jenis template aplikasi seperti Console Application, WPF Application, Windows Form Application, dan masih banyak lagi jenis template project lainnya. Setiap jenis memiliki kelebihan dan kekurangan masing-masing yang disesuaikan dengan kebutuhan pengguna.

Bab 3

Analisis Pemecahan Masalah

3.1 Langkah-langkah Pemecahan Masalah

1. Membuat program menggunakan bahasa C# berorientasi objek. Program terdiri dari beberapa file txt yang digunakan untuk menginisialisasi maze yang akan dipakai program, Lalu membuat program BFS dan DFS yang merupakan solusi penyelesaian permasalahan menggunakan algoritma BFS dan DFS. Serta membuat program tambahan yaitu GUI dari program ini yang akan menampilkan penjelasan hasil pencarian menggunakan algoritma BFS ataupun DFS sesuai dengan yang akan dipilih pengguna nantinya.
2. Membuat GUI berbentuk Windows Form App yang dapat menerima pilihan maze, algoritma yang akan dipilih, dan hasilnya.
3. Mengintegrasikan program dengan GUI agar dapat tervisualisasikan dalam GUI.

3.2 Proses Mapping Persoalan

3.2.1 Proses Mapping Persoalan dengan Algoritma BFS

1. Pertama-tama dibuat beberapa variabel, seperti queue untuk menyimpan node yang akan dieksplorasi, path untuk menyimpan jalur yang telah ditempuh, visited untuk menyimpan node yang telah dikunjungi, tempListPath yaitu menyimpan path dalam satu list. nodeSamePath yaitu menyimpan node mana saja yang menjadi persimpangan ketika mencari satu treasure dengan treasure lainnya.
2. Selanjutnya, dilakukan pencarian menggunakan algoritma BFS dengan memasukkan node awal ke dalam queue dan menandai node tersebut sebagai visited. Selama queue tidak kosong dilakukan dequeue dan dilakukan pengecekan apakah node tersebut merupakan harta karun. Jika ya, maka treasureFoundTuple akan menambahkan tuple tersebut kedalam listnya. Jika semua harta karun telah ditemukan maka proses pencarian akan dihentikan. Selanjutnya, dilakukan pengecekan apakah node yang dieksplorasi merupakan node yang dapat dieksplorasi lebih lanjut. Jika tidak, maka akan dianggap jalan buntu. Program akan mencari node sebelumnya pada jalur yang masih dapat dieksplorasi dan ditambahkan kedalam queue. Lalu melanjutkan pencarian hingga queue kosong.
3. Terakhir, langkah yang dilakukan pada jalur yang ditempuh akan diterjemahkan menjadi urutan langkah dalam bentuk karakter (U untuk up, D untuk down, L untuk left, dan R untuk right) dan dicetak.

Algoritma BFS yang diimplementasikan dalam program ini merupakan salah satu cara untuk menemukan jalur pada sebuah graf dengan cara melalui setiap node secara merata dulu(horizontal) setiap percabangan dari sebuah node. Pada implementasi ini, BFS digunakan untuk menemukan jalur pada sebuah labirin dengan cara menelusuri setiap simpul pada graf sampai menemukan jalur yang tepat menuju setiap harta karun yang ada di dalam labirin.

3.2.2 Proses Mapping Persoalan dengan Algoritma DFS

1. Pertama-tama dibuat beberapa variabel, seperti stack untuk menyimpan node yang akan dieksplorasi, path untuk menyimpan jalur yang telah ditempuh, visited untuk menyimpan node yang telah dikunjungi, dan steps untuk menyimpan langkah-langkah yang telah dilakukan.
2. Selanjutnya, dilakukan pencarian menggunakan algoritma DFS dengan memasukkan node awal ke dalam stack dan menandai node tersebut sebagai visited. Selama stack tidak kosong, dilakukan penghapusan node paling atas dari stack dan dimasukkan ke dalam path. Kemudian, dilakukan pengecekan apakah node tersebut merupakan sebuah harta karun. Jika ya, maka treasureVisited akan bertambah 1. Jika semua harta karun telah ditemukan, pencarian akan dihentikan. Selanjutnya, dilakukan pengecekan apakah node yang dieksplorasi merupakan node yang dapat dieksplorasi lebih lanjut. Jika tidak, maka dianggap sebagai jalan buntu. Program akan mencari node sebelumnya pada jalur yang masih dapat dieksplorasi dan menempatkan node-node tersebut ke dalam stack kembali, kemudian melanjutkan pencarian dari node tersebut.
3. Terakhir, langkah-langkah yang dilakukan pada jalur yang ditempuh akan diterjemahkan menjadi urutan langkah dalam bentuk karakter (U untuk up, D untuk down, L untuk left, dan R untuk right) dan dicetak.

Algoritma DFS yang diimplementasikan dalam program merupakan salah satu cara untuk menemukan jalur pada sebuah graf dengan cara melalui setiap node secara vertikal terlebih dahulu dan menjelajah setiap cabang secara terurut. Pada implementasi ini, DFS digunakan untuk menemukan jalur pada sebuah labirin dengan cara menelusuri setiap simpul pada graf sampai menemukan jalur yang tepat menuju setiap harta karun yang ada di dalam labirin.

3.3 Contoh Ilustrasi Kasus Lain

Ada beberapa kasus maze yang dapat terjadi antara lain :

1. Ketika node K (node awal / Krusty Krab) terdapat lebih dari 1.
2. Ketika tidak ada node T (treasure) sama sekali di map tersebut.

3. Ketika map maze memiliki node selain K,T, dan X.

Untuk handle kasus - kasus tersebut kita telah mengatasinya dengan menggunakan exception sehingga jika menemukan map yang memiliki salah satu atau lebih dari kasus maze diatas kita akan langsung memproses bahwa map tersebut tidak dapat diproses.

Bab 4**Implementasi dan Pengujian****4.1 Implementasi Program Utama**

Program BFS

```
BFS class :  
  
    queue <- new empty Queue of tuple  
  
    treasureFound <- 0  
  
    treasureFoundTuple <- new empty list of tuple  
  
    constructor:  
        initializes queue and treasureFoundTuple to empty lists and  
        treasureFound to 0  
  
    procedure inqueue(maze.getPosition()):  
  
        if node.getLeft() is not Null and notVisited(node.getLeft()) :  
            tuple = (node.getLeft(), 'L')  
            queue.enqueue(tuple)  
  
        if node.getRight() is not Null and notVisited(node.getRight()) :  
            tuple = (node.getRight(), 'R')  
            queue.enqueue(tuple)  
  
        if node.getUp() is not Null and notVisited(node.getUp()) :  
            tuple = (node.getUp(), 'U')  
            queue.enqueue(tuple)
```

```

    if node.getDown() is not Null and notVisited(node.getDown()) :

        tuple = (node.getDown(), 'D')

        queue.enqueue(tuple)

procedure search(maze):

    starti = maze.getStart().getX()

    startj = maze.getStart().getY()

    maze.setPosition(maze.getStart())

    inqueue(maze.getPosition())

    visited.Add(maze.getStart(), 'S')

    while queue.count is not 0 :

        tuple = queue.dequeue()

        visited.add(tuple)

        if(tuple.Item1.getValue() equal to 'T') :

            found = False

            foreach a in treasureFoundTuple :

                if a.Item1 == tuple.Item1 :

                    found = True

                    break

            if not found :

                treasureFoundTuple.Add(tuple)

```

```

        temp = maze.FindNode(tuple.Item1)

        maze.setPosition(temp)

        inqueue(maze.getPosition())

        if(tuple is not in visited) :

            visited.Add(tuple)

List<Tuple<Node, char>> pathTreasure(treasureloc) :

    i = treasureloc.item1.getX()

    j = treasureloc.item1.getY()

    temppath = new list<Tuple<node,char>>()

    visited.reverse()

    found = False

    direction = ''

    while is not found :

        for k = 0 to visited.Count -1 :

            if visited[k].Item1.getX() == i and visited[k].item1.getY()
== j :

                if visited[k].Item1.getValue() equal to 'T' :

                    treasureAdded.Add(visited[k])

                if visited[k].Item1.getValue() equal to 'K' :

                    found = True

                    break

            else :

```

```

        temppath.Add(visited[k])

        direction = visited[k].item2

        switch (direction) :

            case 'U' :

                i = i + 1

                break

            case 'D' :

                i = i - 1

                break

            case 'L' :

                j = j + 1

                break

            case 'R' :

                j = j - 1

                break

        visited.reverse()

        return temppath

procedure findPath(idx) :

    if idx is not -1 :

        if treasureFoundTuple[idx].Item1 is not in treasureAdded :

            if idx + 1 equal to treasureFoundTuple.count :

```

```

        flagNode = new Tuple<Node, char>

        for i = 0 to visited.count - 1 :

            if visited[i] equal to treasureFoundTuple[i] :

                flagnode = visited[i]

                break

        tempListPath.Add(pathTreasure(flagNode))

        findPath(idx - 1)

    else if idx > 0 :

        flagNode = new Tuple<Node, char>

        for i = 0 to visited.count - 1 :

            if visited[i] equal to treasureFoundTuple[i] :

                flagnode = visited[i]

                break

        tempListPath.Add(pathTreasure(flagNode))

        findPath(idx - 1)

    else if idx equal to 0 :

        flagNode = new Tuple<Node, char>

        for i = 0 to visited.count - 1 :

            if visited[i] equal to treasureFoundTuple[i] :

```



```

        flagnode = visited[i]

        break

    tempListPath.Add(pathTreasure(flagNode))

else findPath(idx - 1)

procedure makePath() :

    reverse the order of elements in tempListPath

    for each list a in tempListPath :

        for each tuple element (node, cost) in a :

            for i from 0 to the length of tempListPath - 2 :

                for j from 0 to the length of tempListPath[i] :

                    set cek to false

                    for k from 0 to the length of tempListPath[i + 1] :

                        if tempListPath[i][j].Item1.getX() equals
tempListPath[i + 1][k].Item1.getX() and

                            tempListPath[i][j].Item1.getY() equals
tempListPath[i + 1][k].Item1.getY() then

                                add a new tuple (tempListPath[i][j].Item1, i) to
nodeSamePath

                    set cek to true

```

```

        break out of the innermost loop

        if cek is true then break out of the middle loop

reverse the order of elements in tempListPath

procedure savePath():

    reverse tempListPath

    for i from 0 to size of tempListPath - 1:

        if i == size of tempListPath - 1:

            reverse tempListPath[i]

            cek1 = false

            cek = false

            idxnodesame = -1

            for j from 0 to size of nodeSamePath - 1:

                if nodeSamePath[j].getIndex() == i - 1:

                    cek = true

                    idxnodesame = j

                    break

            if cek:

                for j from 0 to size of tempListPath[i] - 1:

                    if tempListPath[i][j].getX() ==
nodeSamePath[idxnodesame].getX() and tempListPath[i][j].getY() ==
nodeSamePath[idxnodesame].getY():

                        cek1 = true

```

```

        else if cek1:

            path.add(tempListPath[i][j])

        else:

            for j from 0 to size of tempListPath[i] - 1:

                path.add(tempListPath[i][j])

            reverse tempListPath[i]

    else if (i is not equal to 0) :

        tempListPath[i].Reverse()

        for each a in tempListPath[i]

            add a to path

        tempListPath[i].Reverse()

        cek = false

        idxnodesame = -1

        for each a in nodeSamePath

            idxnodesame = idxnodesame + 1

            if a.Item2 is equal to i then

                cek = true

                break

        if cek then

            for j = 1 to tempListPath[i].Count - 1

                if (tempListPath[i][j].Item1.getX() is equal to
nodeSamePath[idxnodesame].Item1.getX()) AND

```

```

        (tempListPath[i][j].Item1.getY() is equal to
nodeSamePath[idxnodesame].Item1.getY()) then

            if tempListPath[i][j - 1].Item2 is equal to
'L' then

                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'R'

                add a to path

            else if tempListPath[i][j - 1].Item2 is
equal to 'R' then

                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'L'

                add a to path

            else if tempListPath[i][j - 1].Item2 is
equal to 'U' then

                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'D'

                add a to path

            else if tempListPath[i][j - 1].Item2 is
equal to 'D' then

                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'U'

                add a to path

            break

        else

```

```

                                if tempListPath[i][j - 1].Item2 is equal to
'L' then

                                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(),   tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'R'

                                add a to path

                                else if tempListPath[i][j - 1].Item2 is
equal to 'R' then

                                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(),   tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'L'

                                add a to path

                                else if tempListPath[i][j - 1].Item2 is
equal to 'U' then

                                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(),   tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'D'

                                add a to path

                                else if tempListPath[i][j - 1].Item2 is
equal to 'D' then

                                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(),   tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'U'

                                add a to path

                                else :

                                if tempListPath[i][j - 1].Item2 is equal to 'L' then

```

```

                                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(),  tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'R'

                                add a to path

                                else if tempListPath[i][j - 1].Item2 is equal to 'R'
then

                                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(),  tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'L'

                                add a to path

                                else if tempListPath[i][j - 1].Item2 is equal to 'U'
then

                                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(),  tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'D'

                                add a to path

                                else if tempListPath[i][j - 1].Item2 is equal to 'D'
then

                                a = Tuple of Node (new
Node(tempListPath[i][j].Item1.getX(),  tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue())) and 'U'

                                add a to path

                                else :

                                WriteLine("i tengah2")

tempListPath[i].Reverse()

cek1 = false

ceka = false

```

```

cekb = false

idxnodesame1 = -1

idxnodesame2 = -1

for a in nodeSamePath:

    idxnodesame1++

    if a.Item2 equals i - 1:

        cek1 = true

        break

for a in nodeSamePath:

    idxnodesame2++

    if a.Item2 equals i:

        cekb = true

        break

if cek1:

    for j in tempListPath[i]:

        if tempListPath[i][j].Item1.getX() equals
nodeSamePath[idxnodesame1].Item1.getX() and
tempListPath[i][j].Item1.getY() equals
nodeSamePath[idxnodesame1].Item1.getY():

            cek1 = true

        else if cek1:

            path.Add(tempListPath[i][j])

else:

```

```

        for j in tempListPath[i]:

            path.Add(tempListPath[i][j])

            WriteLine("x : " + tempListPath[i][j].Item1.getX() + " y : " +
tempListPath[i][j].Item1.getY() + " " + tempListPath[i][j].Item2)

tempListPath[i].Reverse()

if cekb:

    for j in tempListPath[i]:

        if tempListPath[i][j].Item1.getX() equals
nodeSamePath[idxnodesame2].Item1.getX() and
tempListPath[i][j].Item1.getY() equals
nodeSamePath[idxnodesame2].Item1.getY():

            if tempListPath[i][j - 1].Item2 equals 'L':

                a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'R')

                path.Add(a)

            else if tempListPath[i][j - 1].Item2 equals 'R':

                a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'L')

                path.Add(a)

            else if tempListPath[i][j - 1].Item2 equals 'U':

                a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'D')

```



```

        path.Add(a)

        else if tempListPath[i][j - 1].Item2 equals 'D':

            a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'U')

            path.Add(a)

            break

        else:

            if tempListPath[i][j - 1].Item2 equals 'L':

                a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'R')

                path.Add(a)

            else if tempListPath[i][j - 1].Item2 equals 'R':

                a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'L')

                path.Add(a)

            else if tempListPath[i][j - 1].Item2 equals 'U' :

                a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'D')

                path.Add(a)

            else if tempListPath[i][j - 1].Item2 equals 'D' :

                a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'U')

```

```

        path.Add(a)

else

    for j in tempListPath[i]:

        if tempListPath[i][j - 1].Item2 equals 'L':

            a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'R')

            path.Add(a)

        else if tempListPath[i][j - 1].Item2 equals 'R':

            a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'L')

            path.Add(a)

        else if tempListPath[i][j - 1].Item2 equals 'U':

            a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'D')

            path.Add(a)

        else if tempListPath[i][j - 1].Item2 equals 'D':

            a = new Tuple<Node, char>(new
Node(tempListPath[i][j].Item1.getX(), tempListPath[i][j].Item1.getY(),
tempListPath[i][j].Item1.getValue()), 'U')

            path.Add(a)

    tempListPath[i].reverse()

```

```

        if (tempListPath[i][0].Item2 == 'L'):

            Tuple<Node, char> a = new Tuple<Node, char>(new
Node(tempListPath[i][0].Item1.getX(), tempListPath[i][0].Item1.getY() +
1, 'S'), 'R');

            path.Add(a);

        else if (tempListPath[i][0].Item2 == 'R')

            Tuple<Node, char> a = new Tuple<Node, char>(new
Node(tempListPath[i][0].Item1.getX(), tempListPath[i][0].Item1.getY() -
1, 'S'), 'L');

            path.Add(a);

        else if (tempListPath[i][0].Item2 == 'U')

            Tuple<Node, char> a = new Tuple<Node, char>(new
Node(tempListPath[i][0].Item1.getX() + 1,
tempListPath[i][0].Item1.getY(), 'S'), 'D');

            path.Add(a);

        else if (tempListPath[i][0].Item2 == 'D')

            Tuple<Node, char> a = new Tuple<Node, char>(new
Node(tempListPath[i][0].Item1.getX() - 1,
tempListPath[i][0].Item1.getY(), 'S'), 'U');

            path.Add(a);

```

Program DFS

```

DFS class:

    stack <- new empty Stack of nodes

    path <- new empty Stack of nodes

    visited <- new empty List of nodes

```

```
treasureVisited <- 0

steps <- new empty List of characters

constructor:

    initializes stack, path, visited, and steps to empty lists and
treasureVisited to 0

Procedure DFSsearch(graph):

    stack = new Stack<Node>()

    path = new Stack<Node>()

    visited = new List<Node>()

    steps = new List<char>()

    treasureVisited = 0

    stack.Push(graph.getStart())

    visited.Add(graph.getStart())

    while stack is not empty:

        node = stack.Pop()

        path.Push(node)

        visited.Add(node)

        if node.getValue() is 'T':
```

```

        treasureVisited += 1

    if treasureVisited is equal to graph.getTreasure():

        break

    if not exploreable(path.Peek()):

        temp = new List<Node>()

        found = false

        while not found:

            n = path.Pop()

            temp.Add(n)

            if exploreable(n):

                found = true

        for i = temp.Count - 1 to 0:

            path.Push(temp[i])

        for i = 1 to temp.Count - 1:

            path.Push(temp[i])

    if node.getLeft() is not null and notVisited(node.getLeft()):

        temp = graph.FindNode(node.getLeft().getX(),
node.getLeft().getY())

        stack.Push(temp)

```

```

        if node.getUp() is not null and notVisited(node.getUp()):
            temp = graph.FindNode(node.getUp().getX(),
node.getUp().getY())

            stack.Push(temp)

        if node.getRight() is not null and notVisited(node.getRight()):
            temp = graph.FindNode(node.getRight().getX(),
node.getRight().getY())

            stack.Push(temp)

        if node.getDown() is not null and notVisited(node.getDown()):
            temp = graph.FindNode(node.getDown().getX(),
node.getDown().getY())

            stack.Push(temp)

Bool exploreable(node):

    if node.getLeft() is not null and notVisited(node.getLeft()):

        return true

    if node.getUp() is not null and notVisited(node.getUp()):

        return true

    if node.getRight() is not null and notVisited(node.getRight()):

        return true

    if node.getDown() is not null and notVisited(node.getDown()):

```

```

        return true

    return false

Bool notVisited(node):

    for n in visited:

        if n.getX() is equal to node.getX() and n.getY() is equal to
node.getY():

            return false

    return true

Procedure reservePath():

    temp = new Stack<Node>()

    while path.count is not empty :

        temp.push(path.pop())

    path = temp

Procedure makeSteps():

    temp = new Stack<Node>()

    temp = path

    prev = temp.Pop()

    while temp is not empty:

        curr = temp.Pop()

        if curr.getX() is equal to prev.getX() - 1 and curr.getY() is
equal to prev.getY():

```

```
        steps.Add('U')

        else if curr.getX() is equal to prev.getX() + 1 and curr.getY()
is equal to prev.getY():

            steps.Add('D')

        else if curr.getX() is equal to prev.getX() and curr.getY() is
equal to prev.getY() - 1:

            steps.Add('L')

        else if curr.getX() is equal to prev.getX() and curr.getY() is
equal to prev.getY() + 1:

            steps.Add('R')

        prev = curr

List<Node> getPath():

    return path.ToList()

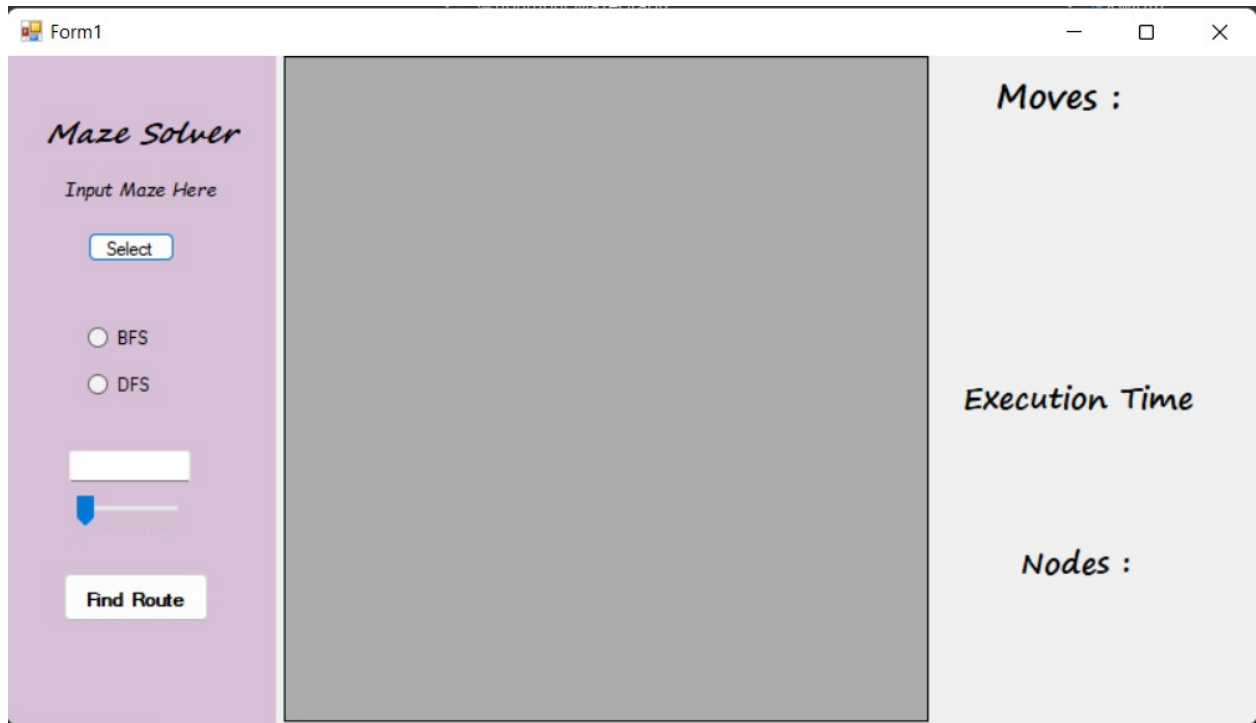
List<char> getSteps():

    return steps
```

4.2 Penjelasan Struktur Data

Struktur data yang digunakan hanyalah bfs dan dfs yang sudah meliputi algoritma searchingnya dan juga penentuan path hasilnya yang disambungkan kepada GUI

4.3 Tata Cara Penggunaan Program



Pertama select file txt maze yang akan digunakan. Lalu Pilih algoritma searching yang akan digunakan antara BFS atau DFS. Lalu program di-run dengan mengclick find route button. User dapat mengatur kecepatan animasi pencarian dengan slider yang terletak diatas find route button.

4.4 Hasil Pengujian

Form1

Maze Solver

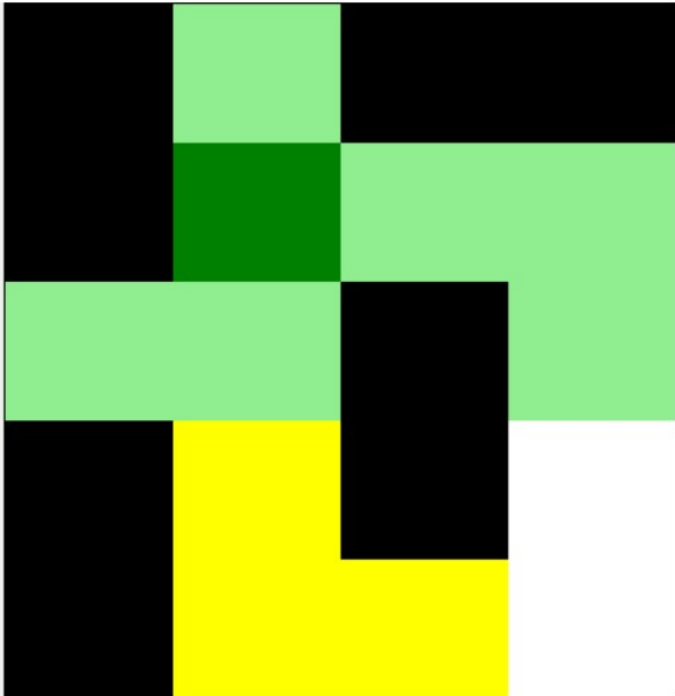
Input Maze Here

Select

☒ BFS
☐ DFS

100

Find Route



Moves :
R-U-U-D-R-R-D-

Execution Time
2 ms

Nodes :
10

Form1

Maze Solver

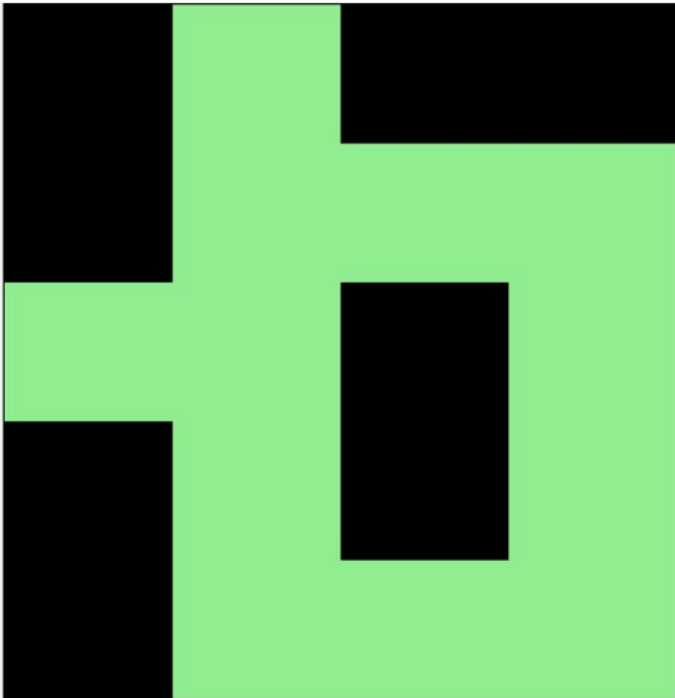
Input Maze Here

Select

☐ BFS
☒ DFS

200

Find Route



Moves :
R-D-D-R-R-U-U-L-L-U-

Execution Time
1 ms

Nodes :
13

4.5 Analisis Desain Solusi

Untuk solusi BFS algoritma yang dibuat terlalu panjang sehingga bisa untuk dibuat lebih simpel lagi. Alternatif algoritma BFS yang bisa digunakan selain yang telah dibuat pada program ini yaitu ketika setiap menemukan treasure maka akan mereset dan mengset treasure tersebut sebagai titik awal dan memulai kembali algoritma BFS.

Bab 5

Kesimpulan dan Saran

5.1 Kesimpulan

Algoritma Breadth-First Search (BFS) dan Depth-First Search dapat digunakan untuk menyelesaikan permainan Maze Treasure. Algoritma DFS menemukan jalur pada sebuah labirin dengan cara menelusuri setiap node secara vertikal atau mendalam, sedangkan algoritma BFS menemukan jalur pada sebuah labirin dengan cara menelusuri secara horizontal atau melebar.

5.2 Saran

Saran-saran yang dapat kami berikan untuk Tugas Besar II IF2211 Strategi Algoritma Semester II 2022/2023 adalah algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS) yang digunakan dalam Tugas Besar ini masih dapat dikembangkan untuk kepentingan efisiensi program karena tentu masih terdapat kekurangan. Selain itu, untuk memudahkan programmer me-maintenance program yang ada, ada baiknya kode program dibuat lebih modular dan tersegmentasi dengan baik. Terakhir, program ini dapat dipublikasikan setelah dikembangkan lebih lanjut agar dapat bermanfaat menjadi referensi publik.

5.3 Refleksi

Setelah menyelesaikan Tugas Besar I IF2211 Strategi Algoritma Semester II 2022/2023, kami menyadari bahwa komunikasi yang terjalin di antara anggota kelompok kami berjalan dengan lancar. Hal ini membuat kami dapat menyelesaikan tugas besar tanpa mengalami kesalahpahaman dan miskomunikasi yang berarti. Sebelum memulai pengerjaan tugas, kami juga melakukan diskusi untuk membahas pembagian tugas bagi setiap anggota kelompok.

5.4 Tanggapan

Tugas Besar Stima kali ini seru dan berkesan dan membuat mahasiswa memahami lebih lanjut tentang algoritma searching BFS dan DFS.

Lampiran

Link Repository : https://github.com/kelvinra/Tubes2_uburubur

Daftar Pustaka

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

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