

Tugas Besar 2 IF2211 Strategi Algoritma

Semester II tahun 2022/2023

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

Disusun oleh :

Ahmad Nadil 13521024/K3

Aulia Mey Diva A. 13521103/K1

M. Abdul Aziz G. 13521128/K2



PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

Daftar Isi

Daftar Isi	2
BAB 1	4
Deskripsi Tugas	4
1.1 Latar Belakang	4
1.2 Deskripsi Tugas	5
1.3 Spesifikasi Program	7
BAB 2	9
Landasan Teori	9
2.1 Graph Traversal	9
2.2 Algoritma Breadth-First Search (BFS)	9
2.3 Algoritma Depth-First Search (DFS)	10
2.4 C# Desktop Application Development	11
BAB 3	13
Analisis Pemecahan Masalah	13
3.1 Langkah-Langkah Pemecahan Masalah	13
3.2 Mapping Persoalan Menjadi Elemen BFS dan DFS	14
3.3 Contoh Ilustrasi Kasus	15
BAB 4	18
Implementasi dan Pengujian	18
4.1 Implementasi Program	18
4.2 Struktur Data dan Spesifikasi Program	22
4.2.1 Attribute	22
4.2.2 Method	23
4.2.3 Constructor	24
4.3 Tata Cara Penggunaan Program	24
4.4 Hasil Pengujian	25
4.5 Analisis Algoritma	35
BAB 5	36
Kesimpulan dan Saran	36
5.1 Kesimpulan	36
5.2 Saran dan Refleksi	36

Daftar Pustaka	37
Link to Repository	37
Link to Figma	37
Link to Youtube	37

BAB 1

Deskripsi Tugas

1.1 Latar Belakang

Tuan Krabs menemukan sebuah labirin distorsi terletak tepat di bawah Krusty Krab bernama El Doremi yang ia yakini mempunyai sejumlah harta karun di dalamnya dan tentu saja ia ingin mengambil harta karunnya. Dikarenakan labirinnya dapat mengalami distorsi, Tuan Krabs harus terus mengukur ukuran dari labirin tersebut. Oleh karena itu, Tuan Krabs banyak menghabiskan tenaga untuk melakukan hal tersebut sehingga ia perlu memikirkan bagaimana caranya agar ia dapat menelusuri labirin ini lalu memperoleh seluruh harta karun dengan mudah.



Gambar 1.1 Labirin di Bawah Krusty Krab

(Sumber: https://static.wikia.nocookie.net/theloudhouse/images/e/ec/Massive_Mustard_Pocket.png/revision/latest?cb=20180826170029)

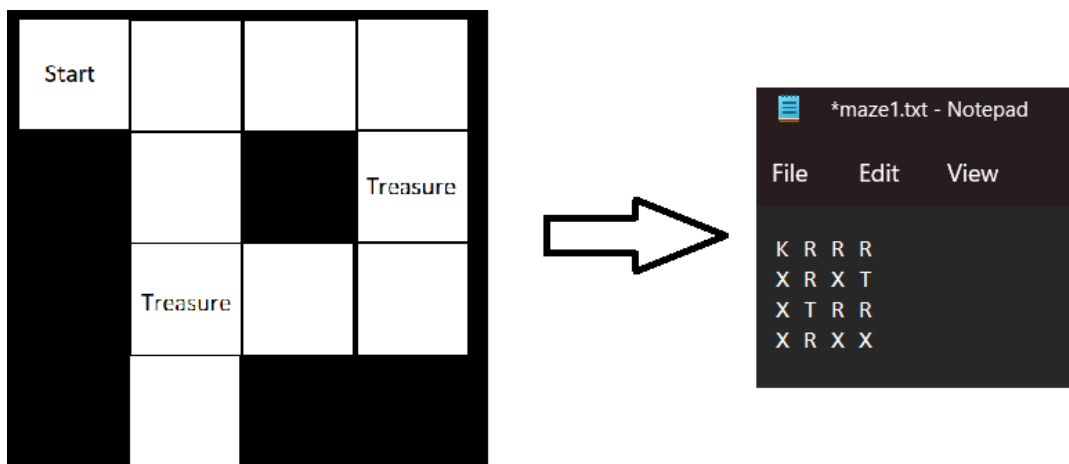
Setelah berpikir cukup lama, Tuan Krabs tiba-tiba mengingat bahwa ketika ia berada pada kelas Strategi Algoritma-nya dulu, ia ingat bahwa ia dulu mempelajari algoritma BFS dan DFS sehingga Tuan Krabs menjadi yakin bahwa persoalan ini dapat diselesaikan menggunakan kedua algoritma tersebut. Akan tetapi, dikarenakan sudah lama tidak menyentuh algoritma, Tuan Krabs telah lupa bagaimana cara untuk menyelesaikan persoalan ini dan Tuan Krabs pun kebingungan. Tidak butuh waktu lama, ia terpikirkan sebuah solusi yang brilian. Solusi tersebut adalah meminta mahasiswa yang saat ini sedang berada pada kelas Strategi Algoritma untuk menyelesaikan permasalahan ini.

1.2 Deskripsi Tugas

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

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

Contoh file input :

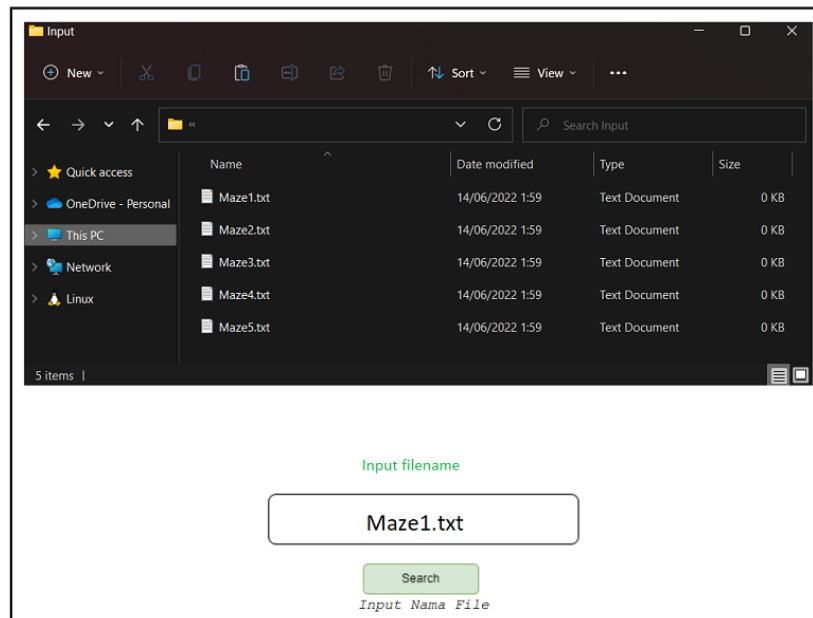


Gambar 1.2 Ilustrasi input file maze

Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), dapat ditelusuri 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. Kami juga diminta untuk memvisualisasikan input txt tersebut menjadi suatu grid maze serta hasil pencarian rute solusinya. Cara

visualisasi grid dibebaskan, sebagai contoh dalam bentuk matriks yang ditampilkan dalam GUI dengan keterangan berupa teks atau warna. Pemilihan warna dan maknanya dibebaskan ke masing - masing kelompok, asalkan dijelaskan di readme / laporan.

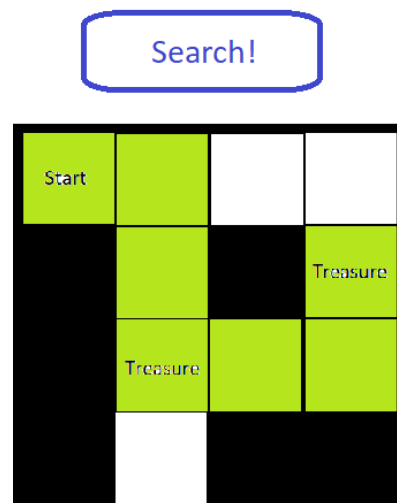
Contoh input aplikasi :



Gambar 1.3 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 meng-handle kasus apabila tidak ditemukan dengan nama file tersebut.

Contoh output Aplikasi :

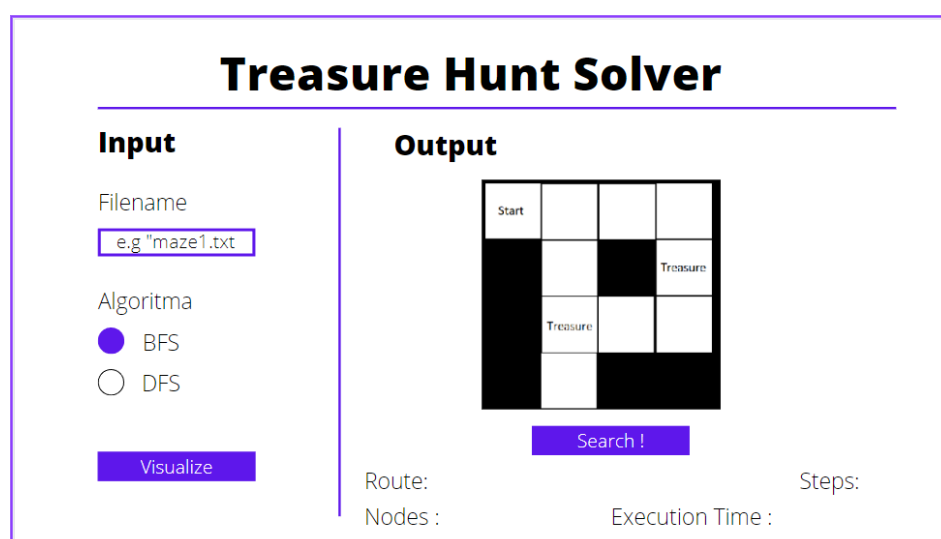


Gambar 1.4 Contoh output program untuk gambar 2

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.

1.3 Spesifikasi Program

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



Gambar 1.5 Tampilan Program Sebelum dicari solusinya

Treasure Hunt Solver

Input
Filename

Algoritma
☒ BFS
☐ DFS

Output

Start			
			Treasure
	Treasure		

Route: R - D - D - R - R - U Steps: 6
Nodes : 11 Execution Time : 850 ms

Gambar 1.6 Tampilan Program setelah dicari solusinya

BAB 2

Landasan Teori

2.1 Graph Traversal

Graph traversal adalah proses mengunjungi semua simpul/sudut dan edge/garis dari sebuah graph yang mengandung banyak simpul dan edge secara sistematis. Tujuannya adalah untuk mengakses atau memanipulasi setiap simpul dalam grafik dengan cara yang efisien dan sistematis. Ada dua jenis graph traversal: Breadth-First Search (BFS) dan Depth-First Search (DFS). Pada BFS, semua simpul di level yang sama dikunjungi sebelum melanjutkan ke level berikutnya sedangkan pada DFS, semua simpul dalam satu jalur dikunjungi terlebih dahulu sebelum memulai jalur berikutnya.

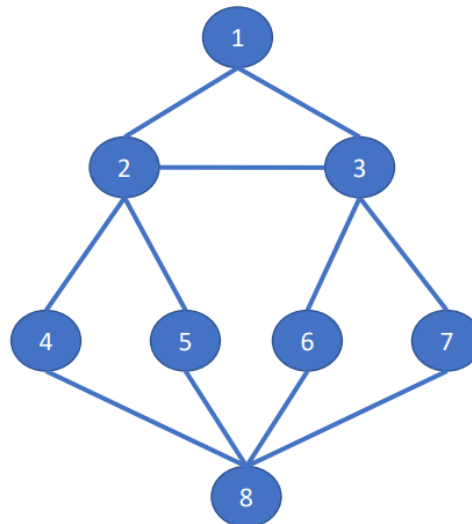


Gambar 2.1.1 Graf Traversal

2.2 Algoritma Breadth-First Search (BFS)

Algoritma Breadth-First Search atau BFS merupakan salah satu algoritma traversal yang memiliki ciri utama pencarian melebar. BFS dan aplikasinya pada graf ditemukan oleh Konrad Zuse pada 1945 pada tesisnya yang tidak dikemukakan, dan ditemukan kembali oleh Edward F. Moore pada 1959 yang mengimplementasikannya untuk mencari jalan tercepat keluar dari sebuah labirin. Pada algoritma ini akan dilakukan eksplorasi semua simpul yang berada

pada level yang sama sebelum melanjutkan ke level berikutnya. Secara umum, algoritma ini dimulai dengan mengunjungi simpul paling awal, lalu menjelajahi simpul yang bertetangga langsung dengan simpul awal tersebut, lalu simpul yang bertetangga dengan simpul tersebut, dan seterusnya hingga mencapai tujuan atau seluruh simpul dalam graf telah dikunjungi.



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

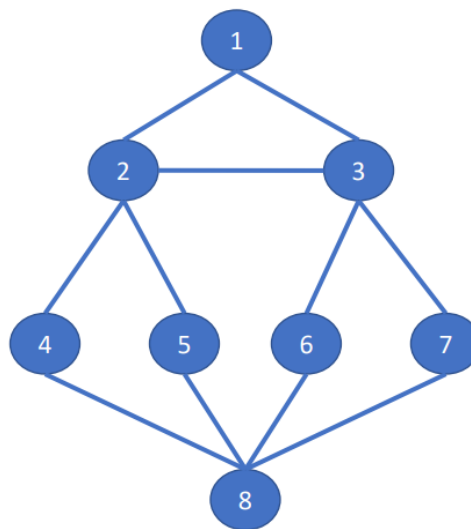
Gambar 2.2.1 Contoh pencarian secara BFS dalam sebuah tree

2.3 Algoritma Depth-First Search (DFS)

Depth-First Search atau DFS merupakan salah satu algoritma traversal graph yang paling umum digunakan dalam ilmu komputer. DFS bekerja dengan menelusuri seluruh simpul atau *node* dalam graf secara mendalam, dengan memulai dari simpul awal kemudian menelusuri simpul yang belum pernah dikunjungi sebelumnya dan terletak pada cabang terdalam yang masih dapat dijangkau. Setelah mencapai ujung cabang, DFS akan kembali ke simpul sebelumnya dan menelusuri cabang lain yang mungkin belum pernah dikunjungi sebelumnya. Algoritma ini menggunakan pendekatan yang sama seperti rekursi dengan melakukan pemanggilan rekursi ke simpul tetangga sampai semua simpul telah dikunjungi.

DFS memiliki kemampuan dan juga kelemahan. DFS memiliki kemampuan untuk mengunjungi setiap simpul dalam graf dengan cara yang sangat

sistematis dan efektif. Di samping itu, DFS memiliki beberapa kelemahan yang perlu dipertimbangkan. Salah satunya dapat menyebabkan masalah ketika algoritma mencari jalan keluar dari sebuah graf yang mengandung siklus karena DFS dapat terjebak dalam loop tak terbatas jika tidak diatur dengan benar. Selain itu, DFS juga dapat memakan waktu yang cukup lama jika digunakan pada graf yang sangat besar atau rumit.



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

Gambar 2.3.1 Contoh pencarian secara DFS dalam sebuah tree

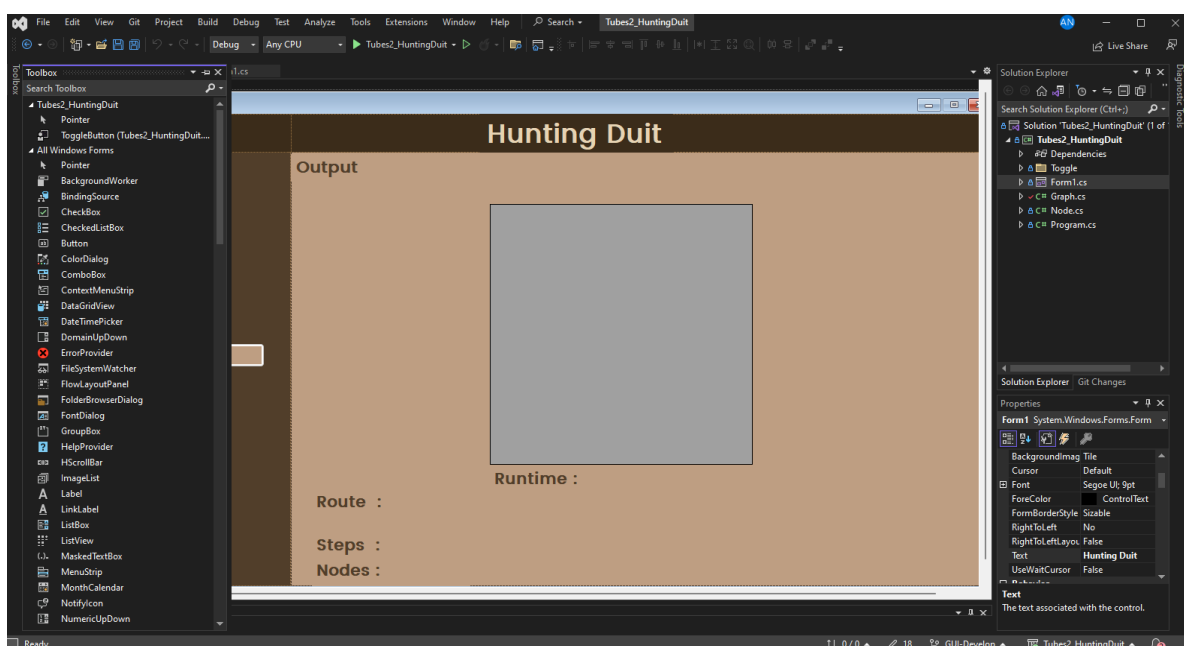
2.4 C# Desktop Application Development

C# atau yang dibaca "See Sharp" merupakan bahasa pemrograman modern yang dikembangkan oleh Microsoft. C# merupakan bahasa yang *Object-Oriented* dan *type-safe* dirancang untuk membuat aplikasi yang berjalan terutama pada platform Windows. C# sangat populer digunakan dalam pengembangan aplikasi desktop, aplikasi web, dan game.

C# dibuat dengan dasar dari keluarga bahasa pemrograman C, sehingga *syntax* yang dimilikinya akan mirip dengan bahasa C, C++, Java, dan JavaScript. Bahasa C# karena merupakan bahasa yang *object-oriented* menerapkan konsep dasar dalam pemrograman *object-oriented* seperti *inheritance*, *encapsulation*, dan *polymorphisme*. C# mengutamakan *versioning* untuk memastikan program dan *library* dapat berkembang secara terus menerus,

dengan salah satu aspek yang digunakannya adalah penggunaan override dan kelas virtual.

C# menggunakan .NET Framework yang merupakan *framework* pengembangan perangkat lunak pada suatu *runtime* yang paling predominan digunakan dalam platform Windows. Framework ini memiliki akses ke *library* kelas yang sangat ekstensif (FCL) dan juga menawarkan *common language runtime* yang menyediakan *virtual machine* sebagai lapisan perlindungan keamanan lebih dan berperan untuk *exception handling* program.



Gambar 2.4.1 Penggunaan bahasa C# dalam Desktop Application Development

Pengembangan aplikasi yang menggunakan bahasa C# biasanya menggunakan IDE Visual Studio. Dapat dilihat dalam gambar 2.4.1, pembuatan Graphical User Interface (GUI) dapat dilakukan dengan cukup mudah, karena hanya perlu melakukan *drag and drop* saja dari *toolbox* yang tersedia. Hal ini sangat memudahkan pengembang dalam melakukan development dikarenakan tidak diperlukannya *hard-code*.

BAB 3

Analisis Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

Permasalahan yang akan diselesaikan dalam Tugas Besar 2 ini adalah pencarian *treasure* dalam sebuah *maze*. *Maze* didapatkan memilih file yang berada pada folder test. Setelah memilih file yang akan dieksekusi, user akan memilih algoritma pemecahan masalah yang sudah disediakan. Kami menggunakan algoritma BFS dan DFS dalam pemecahan masalah tersebut.

File yang dipilih akan diubah menjadi sebuah graf. *Maze* akan direpresentasikan sebagai graf dimana setiap komponen akan disebut sebagai simpul atau *node* dan setiap simpul memiliki simpul tetangga. Kemudian graf diolah untuk menyelesaikan masalah menggunakan algoritma traversal graf. Implementasi BFS menggunakan metode iteratif dan struktur data *queue* sedangkan DFS menggunakan metode rekursif dan struktur data *stack*. Saat sedang melakukan pencarian dengan algoritma BFS dan DFS, program akan menghitung *treasure* yang sudah ditemukan. Bila *treasure* yang ditemukan jumlahnya sama dengan jumlah *treasure* yang berada dalam graf tersebut, maka program akan menghentikan iterasi.

Program juga diimplementasikan menggunakan Graphical User Interface (GUI) untuk melakukan visualisasi proses pencarian *treasure* dalam bentuk *maze* atau labirin. GUI juga meminta input dari user untuk memilih file dalam komputer serta memilih algoritma yang diinginkan untuk menyelesaikan permasalahan tersebut. Nantinya setelah program selesai mengeksekusi, program akan mengeluarkan *output* berupa waktu eksekusi, rute, jumlah simpul yang di cek, dan jumlah langkah. GUI diimplementasikan menggunakan bahasa pemrograman C# yang merupakan sebuah *tools* untuk mengembangkan aplikasi desktop yang dilengkapi dengan .NET *framework*. Pembuatan GUI menggunakan Visual Studio dengan menggunakan berbagai macam library bawaan C# dan juga .NET serta menggunakan WinForm sebagai *platform* pengembangan aplikasi.

3.2 Mapping Persoalan Menjadi Elemen BFS dan DFS

Mapping elemen-elemen dari algoritma yang akan dibuat berdasarkan pemecahan masalah tersebut adalah :

- a. *Krusty Krab* (Disimbolkan "K" dalam peta) : Simpul graf
- b. Lintasan (Disimbolkan "R" dalam peta) : Simpul graf
- c. *Treasure* (Disimbolkan "T" dalam peta) : Simpul graf

Simpul dalam graf tersebut dikatakan bertetangga apabila letaknya bersebelahan dalam peta maze. Informasi mengenai ketetanggaan setiap simpul tersebut disimpan dalam *adjacency matrix*.

Dengan elemen-elemen tersebut, rancangan singkat algoritma BFS dan DFS untuk menyelesaikan permasalahan ini yaitu sebagai berikut :

- a. Algoritma BFS menggunakan struktur data *queue* untuk menyimpan simpul-simpul yang akan dikunjungi berikutnya dan menggunakan metode iteratif untuk mengiterasi simpul-simpul tetangga yang akan dikunjungi berikutnya. *Krusty Krab* akan di-*enqueue* sebagai simpul pertama kali yang akan dikunjungi. Kemudian, simpul-simpul yang bertetangga (lintasan atau *treasure*) dengan simpul yang sedang dikunjungi ini akan di-*enqueue* ke dalam *queue nodes*. Iterasi awal dari simpul *Krusty Krab* selesai dan dilanjutkan pengunjungan ke simpul tetangga dengan men-*dequeue queue nodes*. Simpul tersebut kemudian dilakukan pengecekan apakah merupakan *treasure* atau bukan. Apabila simpul yang sedang dikunjungi bukan merupakan simpul *treasure* maka akan diulangi langkah *enqueue* tetangga dan *dequeue queue nodes* hingga menemukan *treasure*. Setelah menemukan *treasure*, dicek apabila masih terdapat *treasure* yang belum ditemukan, akan dilakukan proses BFS seperti langkah-langkah sebelumnya namun bedanya, simpul pertama yang di-*enqueue* bukanlah *Krusty Krab* melainkan simpul dimana *treasure* terakhir kali ditemukan. Proses ini akan berulang hingga ditemukan semua *treasure* dalam maze.
- b. Algoritma DFS menggunakan struktur data *stack* untuk menyimpan simpul-simpul yang akan dikunjungi berikutnya dan menggunakan metode rekursif. Simpul *krusty krab* di-*push* ke dalam *stack*. *Stack* akan di-*pop* kemudian simpul-simpul yang bertetangga terhadap simpul hasil *pop* tersebut di-*push* ke dalam *stack nodes* Langkah sebelumnya akan diulangi lagi dimana *stack* akan di-*pop* kemudian dicek apakah simpul

dari *pop* adalah *treasure*. Jika simpul bukanlah *treasure* maka simpul tetangga yang bisa dikunjungi akan dikunjungi kemudian proses *push* dan *pop* akan berlanjut hingga ditemukan *treasure*. Lalu apabila tidak ada simpul yang belum dikunjungi, maka akan dilakukan *backtracking* dan mengunjungi simpul sebelumnya. Jika semua *treasure* belum ditemukan, maka pencarian *treasure* dengan DFS akan terus dilanjutkan hingga dihasilkan rute dimana semua *treasure* dapat ditemukan.

3.3 Contoh Ilustrasi Kasus

Untuk memperjelas analisis pemecahan masalah, kami akan melakukan ilustrasi kasus dengan penyelesaiannya. Misal pada kasus dimana map berbentuk sebagai berikut :

Treasure Hunt Solver

Input

Filename

Algoritma
☒ BFS
☐ DFS

Output

Start			
			Treasure
	Treasure		

Route:
Nodes :

Steps:
Execution Time :

Gambar 3.3.1 Ilustrasi Kasus

Pada pemecahan kasus menggunakan metode DFS, elemen yang menjadi start dari maze akan dimasukkan ke dalam *list of Node path*. Setelah itu, elemen tersebut akan dicek ketetanggaannya. Bila tetangganya belum pernah dikunjungi, node dari tetangga akan di *enqueue* ke dalam *queue*. Kemudian simpul akan di cek apakah dia merupakan *node treasure*. Kemudian program akan berjalan secara iteratif dengan menggunakan node yang merupakan Top of Queue saat itu. Node tersebut akan dicek node ketetanggaannya seperti node sebelumnya. Bila tidak ada lagi node ketetanggan yang belum dikunjungi,

program akan melanjutkan proses iterasi sesuai antrian dalam queue. Pemecahan kasus tersebut kami tuangkan dalam bentuk tabel sebagai berikut :

TABEL BFS

HEAD	QUEUE	STATUS
Node 0 (start)	[Node 1]	TREASURE = 0
Node 1	[Node 2, Node 5]	TREASURE = 0
Node 2	[Node 5, Node 3]	TREASURE = 0
Node 5	[Node 3, Node 9]	TREASURE = 0
Node 3	[Node 9, Node 7]	TREASURE = 0
Node 9	[Node 7, Node 10, Node 13]	TREASURE = 1
Node 7	[Node 10, Node 13, Node 11]	TREASURE = 2 (Program Berhenti)

Pada pemecahan kasus menggunakan metode DFS, elemen yang menjadi start akan dimasukkan ke dalam *list of Node*. Setelah itu, elemen tersebut akan dicek ketetanggaannya. Bila tetanggaanya belum pernah dikunjungi, *node* dari tetangga akan di *push* ke dalam *stack*. Kemudian simpul akan di cek apakah dia merupakan *node treasure*. Kemudian program akan berjalan secara rekursif dengan menggunakan *node* yang merupakan *Top of Stack* saat itu. *Node* tersebut akan dicek *node* ketetanggaannya seperti *node* start sebelumnya. Bila tidak ada lagi *node* ketetanggaannya yang belum dikunjungi sama sekali, program akan melakukan *backtrack* sampai menemukan *node* yang memiliki tetangga yang belum pernah dikunjungi. Pemecahan kasus tersebut kami tuangkan dalam bentuk tabel seperti berikut :

TABEL DFS

HEAD	STACK	STATUS
Node 0 (start)	[Node 1]	TREASURE = 0

Node 1	[Node 5, Node 2]	TREASURE = 0
Node 5	[Node 9, Node 2]	TREASURE = 0
Node 9	[Node 13, Node 10, Node 2]	TREASURE = 1
Node 13	[Node 10, Node 2]	TREASURE = 1
Node 9	[Node 10, Node 2]	BACKTRACK TREASURE = 1
Node 10	[Node 11, Node 2]	TREASURE = 1
Node 11	[Node 7, Node 2]	TREASURE = 1
Node 7	[Node 3, Node 2]	TREASURE = 2 (Program Berhenti)

BAB 4

Implementasi dan Pengujian

4.1 Implementasi Program

a. BFS

```
Function BFS() -> List of Node
{ Mencari urutan simpul-simpul yang harus dikunjungi dari titik
awal dalam graf untuk meraih seluruh treasure }
```

Kamus Lokal

```
start : Node
treasure_list : List of Integer
result : list of Node
pathTemp : list of Node
```

Function BFSHelper(start : Node, goal : Integer) -> List of Node
{ Diberikan simpul awal dan nilai spesifik suatu treasure, akan dicari lintasannya dengan metode BFS }

Algoritma

```
start <- nodes.Find(x => x.isStart);
treasure_list <- treasures();
result <- new List<Node>();
pathTemp <- new List<Node>();
result.Add(start);
foreach(int treasure in treasure_list){
    result.AddRange(BFSHelper(start, treasure));
    start <- result[result.Count - 1];
}
-> result;
```

```
Function BFSHelper(start : Node, goal : Integer ) -> List of Node
{ Diberikan simpul awal dan nilai spesifik suatu treasure, akan dicari lintasannya dengan metode BFS }
```

Kamus Lokal

```
path : Queue of List of Node
pathTemp : List of Node
newPath : List of Node
LastNode : Node
```

Algoritma

```
path <- new Queue<List<Node>>();
path.Enqueue(new List<Node> { start });
```

```

{ while loop until the first element in the queue contains the
goal }

while (path.Count > 0)
{
{ dequeue the first element in the queue }
pathTemp <- path.Dequeue();
lastNode <- pathTemp[pathTemp.Count - 1];
System.Console.WriteLine(lastNode.val);

{ if the last node in the path is the goal, return the path }
if (lastNode.val = goal)
{
    pathTemp.RemoveAt(0);
    -> pathTemp;
}

{ if the last node in the path is not the goal, enqueue all the
neighbors of the last node }
foreach (neighbor in adjList[lastNode])
{
    if (!pathTemp.Contains(neighbor))
    {
        newPath <- new List<Node>(pathTemp);
        newPath.Add(neighbor);
        path.Enqueue(newPath);
    }
}
}
}

```

Function TSPBFS(result : List of Node) -> List of Node
{ Mencari urutan simpul-simpul yang harus dikunjungi dari titik treasure terakhir ditemukan dalam graf untuk dapat menuju titik awal }

Kamus Lokal

-

Algoritma

```

result.AddRange(BFSHelper(result[result.Count - 1], nodes.Find(x
=> x.isStart).val));
-> result;

```

b. DFS

Function DFS(ctr : Integer, awal : Node , tc : List of Integer, visitedNode : List of Int, visual : List of Int, res : List of Node, simpule : Stack of Node) -> Tuple of List of Node and List of Integer
{ Fungsi ini melakukan DFS pada graf dan mengembalikan nilai

tuple yang berisi pasangan list of node dan list of integer }

Kamus Lokal

top : Node
init, notvisited : Boolean
temp, hasil : Node
bt, idx, size , hai, count : Integer

Algoritma

```
if(tc.Count == 0){
    { basis dan kalau stack belum kosong }
    -> new Tuple<List<Node>, List<int>> (res, visual);
} else{
    res.Add(awal);
    visual.Add(0);
    visitedNode[awal.val] <- 1;
    nodedfschecked = nodedfschecked + 1;
    if(awal.isStart){
        for(int i=0; i<adjList[awal].Count; i++){
            { push adjacency of the first elemen }
            simpulE.Push(adjList[awal][i]);
        }
        -> DFS(ctr, simpulE.Peek(), tc, visitedNode, visual, res,
simpulE);
    } else {
        top <- simpulE.Pop();
        count <- 0;
        init <- false, notvisited = false;
        for(int i<-0; i<adjList[awal].Count; i++){
            { push adjacency of the first elemen }
            { if it has not visited before }
            if(visitedNode[adjList[awal][i].val] /= 1){
                simpulE.Push(adjList[awal][i]);
                count = count + 1;
            }
            init <- true;
        }
        if(awal.isTreasure){
            tc <- removeTreasure(tc, awal);
        }
        { gaada lagi yang bisa dikunjungin, BACKTRACK }
        temp <- res[res.Count-1];
        if (count == 0 && init == true && (tc.Count != 0)){
            idx <- 2;
            size <- res.Count;
            while(!notvisited){
                hasil <- res[size-idx];
                res.Add(hasil);
                visual.Add(1);
                for (int j<-0; j<adjList[hasil].Count; j++){
                    hai <- adjList[hasil][j].val;
                    if (visitedNode[hai] = 0){
                        notvisited <- true;
                        temp <- adjList[hasil][j];
                        break;
                    }
                }
            }
        }
    }
}
```

```

        }
    }
    idx <- idx + 1;
}

if(notvisited && tc.Count != 0){
    -> DFS(ctr, temp, tc, visitedNode, visual, res,
simpule);
} else{
    if(tc.Count != 0){
        -> DFS(ctr, simpule.Peek(), tc, visitedNode,
visual, res, simpule);
    } else {
        return new Tuple<List<Node>, List<int>>> (res,
visual);
    }
}
}
}

```

Function TSPDFS(ctr : Integer, awal : Node , tc : List of Integer, visitedNode : List of Int, visual : List of Int, res : List of Node, simpule : Stack of Node) -> Tuple of List of Node and List of Integer
 { Fungsi ini melakukan DFS pada graf dan mengembalikan nilai tuple yang berisi pasangan list of node dan list of integer }

Kamus Lokal

top : Node
 init, notvisited : Boolean
 temp, hasil : Node
 bt, idx, size , hai, count : Integer

Algoritma

```

if(awal.isStart){
    { add start to list results
    base of recursive }
    res.Add(awal);
    -> new Tuple<List<Node>, List<int>>> (res, visual);
} else{
    { add node to result and flag it to visited }
    res.Add(awal);
    visual.Add(0);
    visitedNode[awal.val] <- 1;
    nodedfschecked <- nodedfschecked + 1;

    count <- 0;
    init <- false, notvisited <- false;

    for(int i<-adjList[awal].Count-1; i>=0; i--){

```

```

        { push adjacency of the first elemen
        if it has not visited before }
        if (visitedNode[adjList[awal][i].val] != 1){
            simpulE.Push(adjList[awal][i]);
            count <- count + 1;
        }
        init <- true;
    }

    { backtrack if there is no other node that haven't been
visited }
    temp <- res[res.Count-1];
    if (count = 0 & init = true){
        idx <- 2;
        size <- res.Count;

        while(!notvisited){
            hasil <- res[size-idx];
            res.Add(hasil);
            visual.Add(1);
            { check if the adjacent nodes have been visited
before }
            for (int j=0; j<adjList[hasil].Count; j++){
                hai <- adjList[hasil][j].val;
                if (visitedNode[hai] = 0){
                    notvisited <- true;
                    temp <- adjList[hasil][j];
                    break;
                }
            }
            idx = idx + 1;
        }
    }
    { recursive }
    if (notvisited){
        -> TSPDFS(ctr, temp, tc, visitedNode, visual, res,
simpulE);
    } else{
        -> TSPDFS(ctr, simpulE.Peek(), tc, visitedNode, visual,
res, simpulE);
    }
}

```

4.2 Struktur Data dan Spesifikasi Program

4.2.1 Attribute

a. class Node

- int val : digunakan untuk mengakses value dari node

- bool isStart : digunakan untuk mengecek apakah node tersebut adalah *start node*.
- bool isTreasure : digunakan untuk mengecek apakah node tersebut adalah *treasure*.
- bool visited : digunakan untuk mengecek apakah node tersebut sudah pernah dikunjungi.

b. class Graph

- List<Node> nodes : list berisi node dari map file input
- Dictionary<Node, List<Node>> adjList : pair dimana value List<Node> merupakan list tetangga yang dimiliki oleh key node.
- int treasureCount : merupakan atribut penghitung jumlah treasure yang ada di dalam maze
- int nodedfschecked : merupakan atribut untuk menghitung jumlah node yang di check

4.2.2 Method

a. class Graph

- public void makeGraph : menerima parameter input string berupa nama file. Prosedur ini digunakan untuk mengubah file menjadi bentuk graf.
- public void AddNode : menerima parameter input integer val, boolean isStart dan boolean isTreasure. Prosedur ini digunakan untuk menambahkan node pada graf.
- public void AddEdge : menerima parameter input integer val, boolean isStart dan boolean isTreasure. Prosedur ini digunakan untuk menambahkan sisi pada graf.
- public void PrintGraph : prosedur ini digunakan untuk mem=ncetak graf ke layar. Hal ini digunakan untuk pengecekan graf.
- public List<int> way : Fungsi ini digunakan untuk mendapatkan path. Fungsi akan mereturn list of integer berisi path.
- public List<int> treasure : Fungsi ini digunakan untuk mendapatkan nilai simpul yang memiliki treasure.
- public List<int> removeTreasure : Fungsi ini digunakan untuk menghapus value node yang memiliki treasure yang sudah ditemukan oleh DFS.

- `public Tuple<List<Node>, List<int>>> DFS` : Fungsi ini digunakan untuk menyelesaikan kasus yang diberikan file input dan akan mereturn hasil path yang ditemukan secara DFS. Fungsi ini menggunakan metode recursive dengan basis treasure yang ada sudah di visit semua.
- `public Tuple<List<Node>, List<int>>> TSPDFS` : Fungsi ini digunakan untuk menyelesaikan kasus yang diberikan file input dan akan mereturn hasil path yang ditemukan dengan menggunakan pemecahan *travelling salesman problem* secara DFS. Fungsi ini menggunakan metode *recursive* dengan basis apabila node yang merupakan start sudah dikunjungi kembali
- `.public List<Node> BFSHelper` : Fungsi ini digunakan untuk membantu fungsi BFS.
- `public List<Node> BFS` : Fungsi ini digunakan untuk menyelesaikan kasus yang diberikan file input dan akan mereturn hasil path yang ditemukan secara BFS. Fungsi ini menggunakan metode iteratif dan akan berhenti mengeksekusi ketika jumlah path nol.
- `public List<Node> TSPBFS` : Fungsi ini digunakan untuk menyelesaikan kasus yang diberikan file input dan akan mereturn hasil path yang ditemukan dengan menggunakan pemecahan *travelling salesman problem* secara BFS. Fungsi ini menggunakan metode iteratif.

4.2.3 Constructor

- a. `class Node`
 - `public Node` : menerima input parameter `int val`, `bool isStart`, dan `bool isTreasure`. *Constructor* digunakan untuk menginisiasi `Node`.
- b. `class Graph`
 - `public Graph` : digunakan untuk menginisiasi `Graph`.

4.3 Tata Cara Penggunaan Program

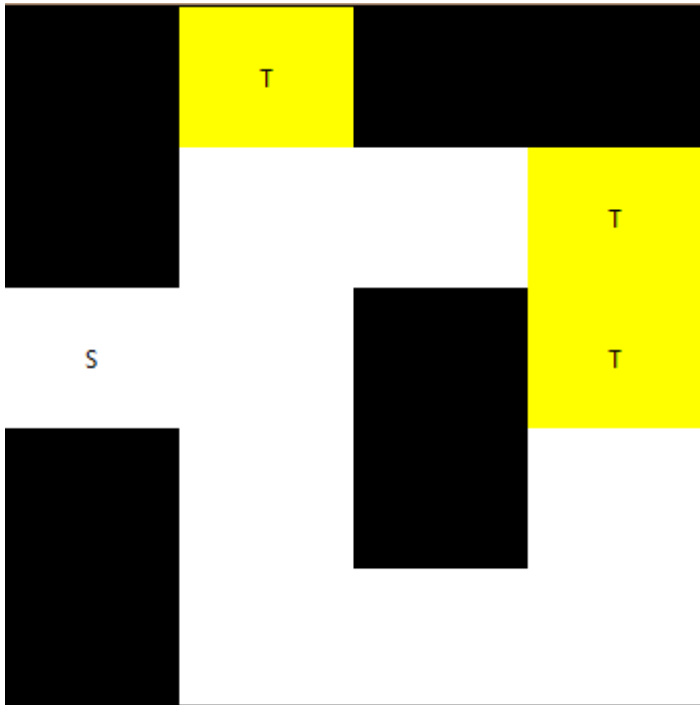
Untuk mem-*build* program, diperlukan langkah seperti berikut :

1. Buka terminal pada device anda.
2. Ketik `cd src`.
3. Ketik `dotnet run`.

Setelah build program berhasil, program dapat dimulai dengan melakukan dengan cara sebagai berikut :

1. Menekan tombol "Choose File" dan memilih file input yang ingin digunakan.
2. Pilih algoritma yang ingin digunakan untuk menyelesaikan kasus.
3. Pilih waktu delay yang diinginkan.
4. Tekan tombol "Search".
5. Program akan mulai mengeksekusi dan program memunculkan visualisasi dari pengeksekusian.
6. Setelah program selesai mengeksekusi, program akan menampilkan rute yang didapatkan, jumlah langkah, jumlah node yang dicek, dan waktu eksekusi.

4.4 Hasil Pengujian

TEST 1	
	
ALGORITMA	HASIL

BFS

Hunting Duit

Input

Algorithm

- ☒ BFS
- ☐ DFS
- ☐ TSP

Filename : sampel-1.txt

Choose File...

Output Delay

103 ms

Search

Output

Runtime : 0,0126 ms

Route : R U U D R R D

Steps : 8

Nodes : 7

TSP BFS

Hunting Duit

Input

Algorithm

- ☐ BFS
- ☐ DFS
- ☒ TSP

Filename : sampel-1.txt

Choose File...

Output Delay

100 ms

Search

Output

Runtime : 0,016 ms

Route : R U U D R R D U L L D L

Steps : 13

Nodes : 7

DFS

Hunting Duit

Input

Algorithm

- ☐ BFS
- ☒ DFS
- ☐ TSP

Filename : sampel-1.txt

Choose File...

Output Delay

100 ms

Search

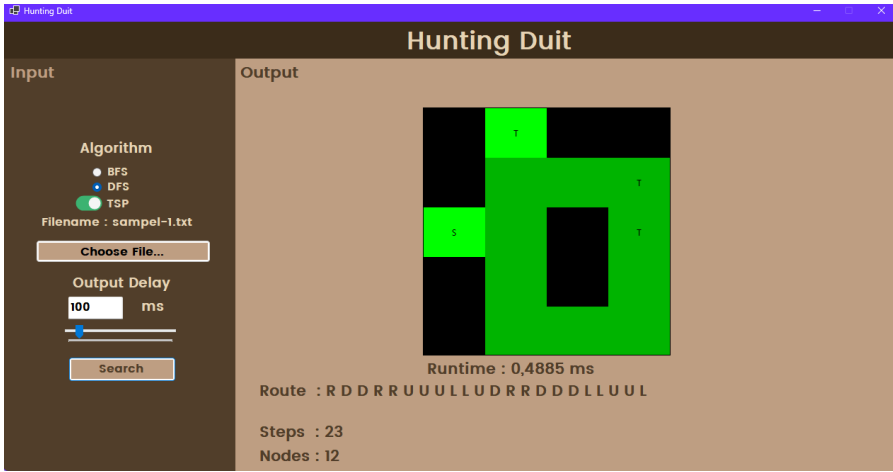
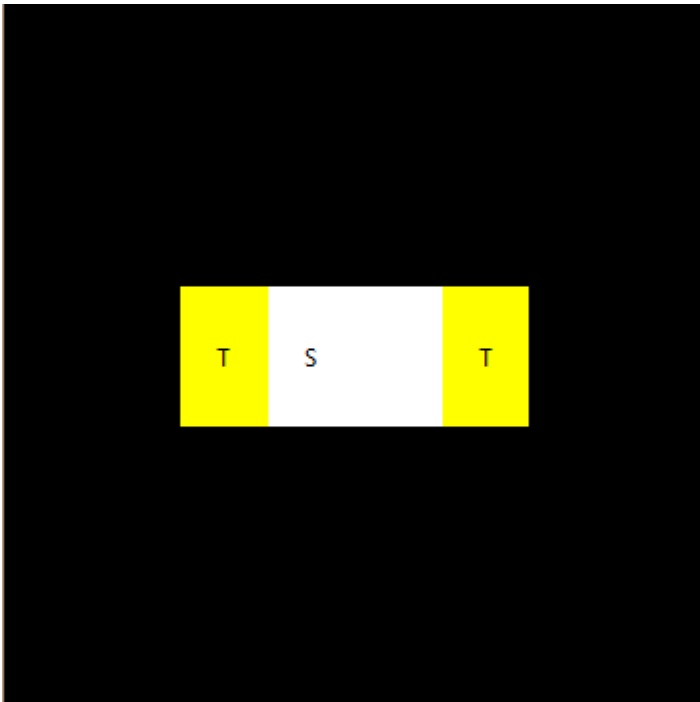
Output

Runtime : 0,7973 ms

Route : R D D R R U U L L U

Steps : 12

Nodes : 12

<p>TSP DFS</p>	
<p>TEST 2</p>	
	
<p>ALGORITMA</p>	<p>HASIL</p>

BFS

Hunting Duit

Input

Algorithm

- ☒ BFS
- ☐ DFS
- ☐ TSP

Filename : sampel-2.txt

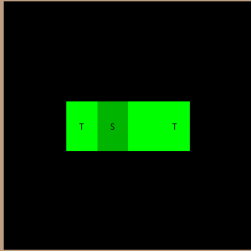
Choose File...

Output Delay

100 ms

Search

Output



Runtime : 0,0273 ms

Route : L R R R

Steps : 5

Nodes : 4

TSP BFS

Hunting Duit

Input

Algorithm

- ☐ BFS
- ☐ DFS
- ☒ TSP

Filename : sampel-2.txt

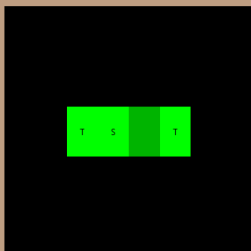
Choose File...

Output Delay

100 ms

Search

Output



Runtime : 0,3205 ms

Route : L R R R L L

Steps : 7

Nodes : 4

DFS

Hunting Duit

Input

Algorithm

- ☐ BFS
- ☒ DFS
- ☐ TSP

Filename : sampel-2.txt

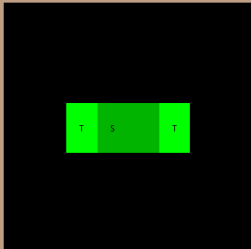
Choose File...

Output Delay

100 ms

Search

Output



Runtime : 0,0081 ms

Route : R R L L L

Steps : 6

Nodes : 4

TSP DFS



TEST 3

S	T		
	T		
	T		
	T		
	T		
	T		
	T		
	T		
	T		
	T		

ALGORITMA

HASIL

BFS

Hunting Duit

Input

Algorithm

- ☒ BFS
- ☐ DFS
- ☐ TSP

Filename : sampel-5.txt

Choose File...

Output Delay

100 ms

Search

Output

Runtime : 0,023 ms

Route : R D D D D D D D D D

Steps : 12

Nodes : 12

TSP BFS

Hunting Duit

Input

Algorithm

- ☐ BFS
- ☐ DFS
- ☒ TSP

Filename : sampel-5.txt

Choose File...

Output Delay

100 ms

Search

Output

Runtime : 0,2113 ms

Route : R D D D D D D D D D U U U U U U U U U U U L

Steps : 23

Nodes : 12

DFS

Hunting Duit

Input

Algorithm

- ☐ BFS
- ☒ DFS
- ☐ TSP

Filename : sampel-5.txt

Choose File...

Output Delay

100 ms

Search

Output

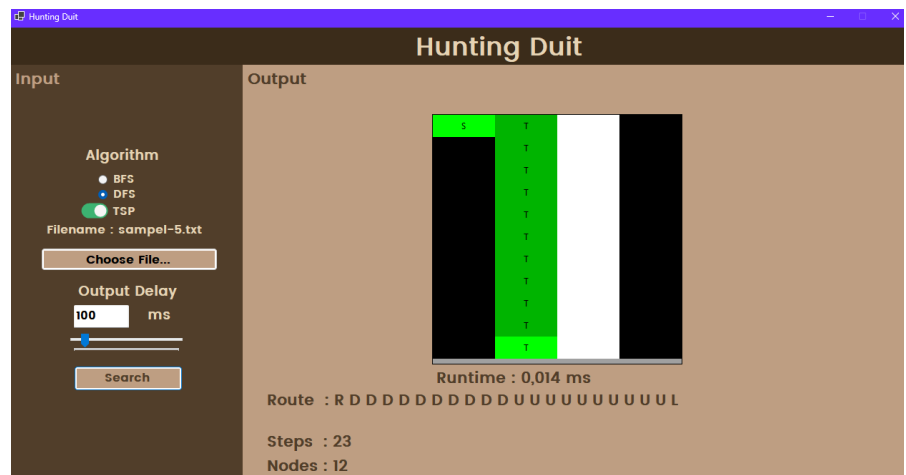
Runtime : 0,012 ms

Route : R D D D D D D D D D

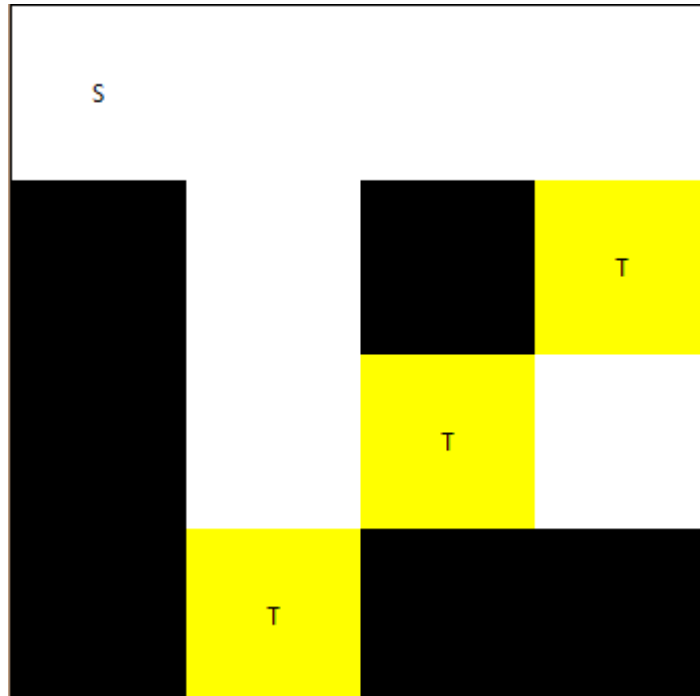
Steps : 12

Nodes : 12

TSP DFS



TEST 4



ALGORITMA

HASIL

BFS

Hunting Duit

Input

Algorithm

- ☒ BFS
- ☐ DFS
- ☐ TSP

Filename : maze1.txt

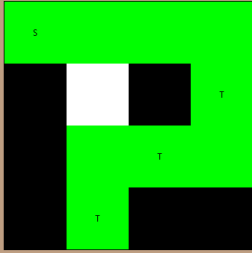
Choose File...

Output Delay

100 ms

Search

Output



Runtime : 0,0223 ms

Route : R R R D D L L D

Steps : 9

Nodes : 9

TSP BFS

Hunting Duit

Input

Algorithm

- ☒ BFS
- ☐ DFS
- ☐ TSP

Filename : maze1.txt

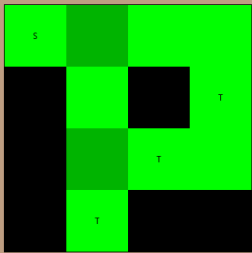
Choose File...

Output Delay

100 ms

Search

Output



Runtime : 0,0304 ms

Route : R R R D D L L D U U U L

Steps : 13

Nodes : 10

DFS

Hunting Duit

Input

Algorithm

- ☐ BFS
- ☒ DFS
- ☐ TSP

Filename : maze1.txt

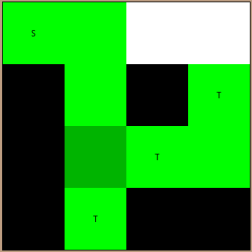
Choose File...

Output Delay

100 ms

Search

Output



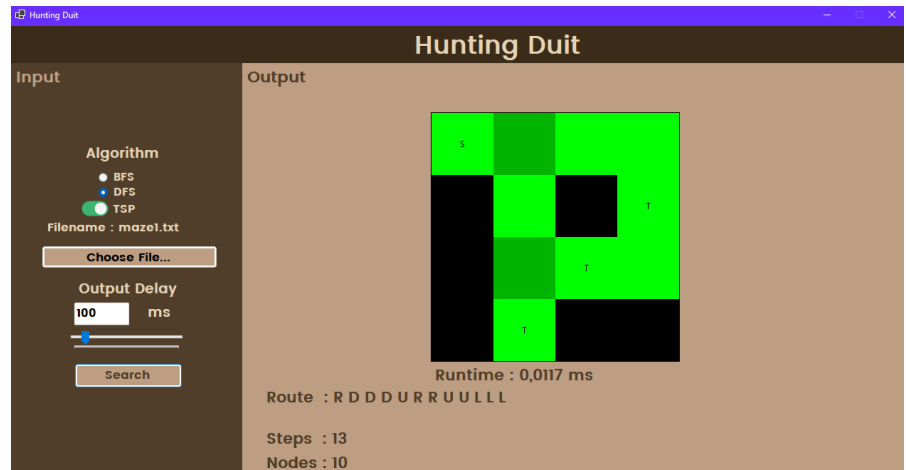
Runtime : 0,0102 ms

Route : R D D D U R R U

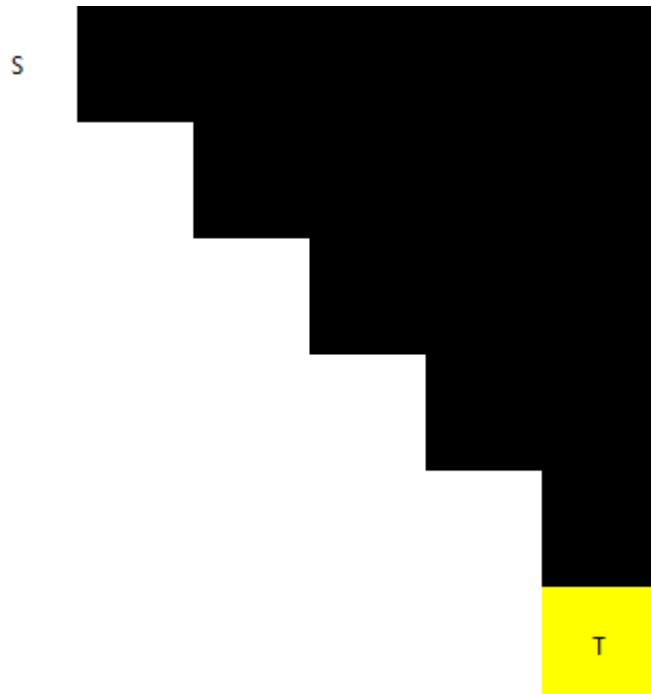
Steps : 9

Nodes : 8

TSP DFS



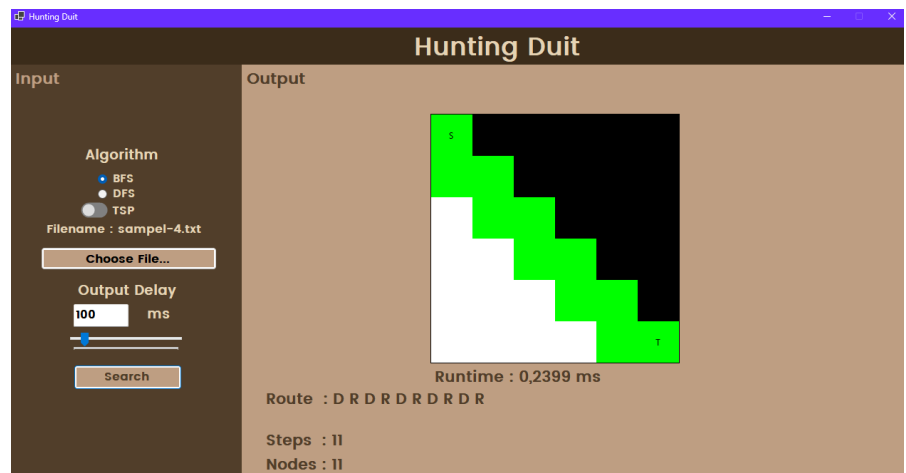
TEST 5



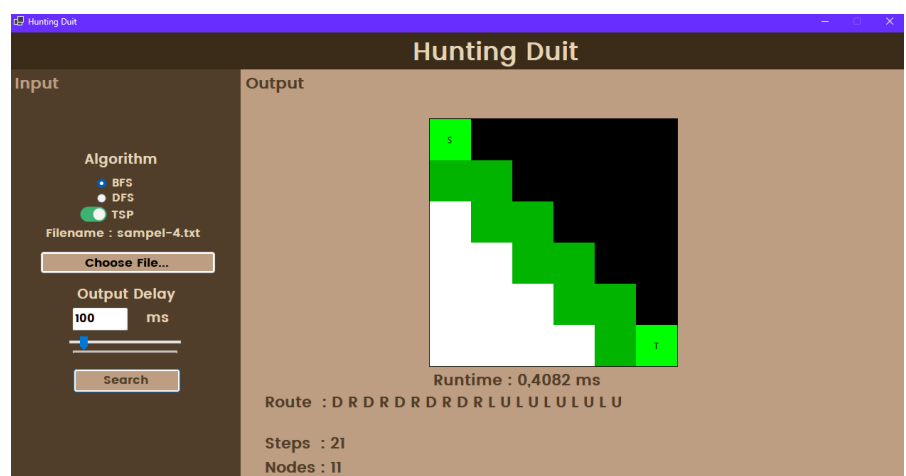
ALGORITMA

HASIL

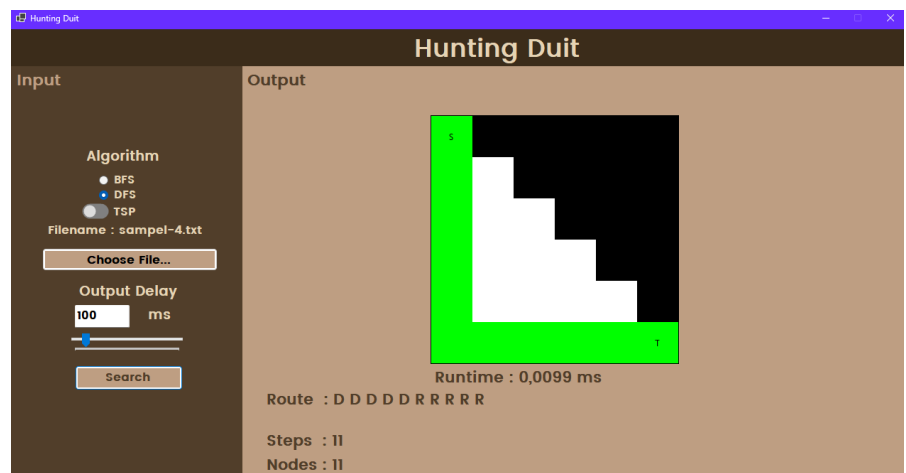
BFS



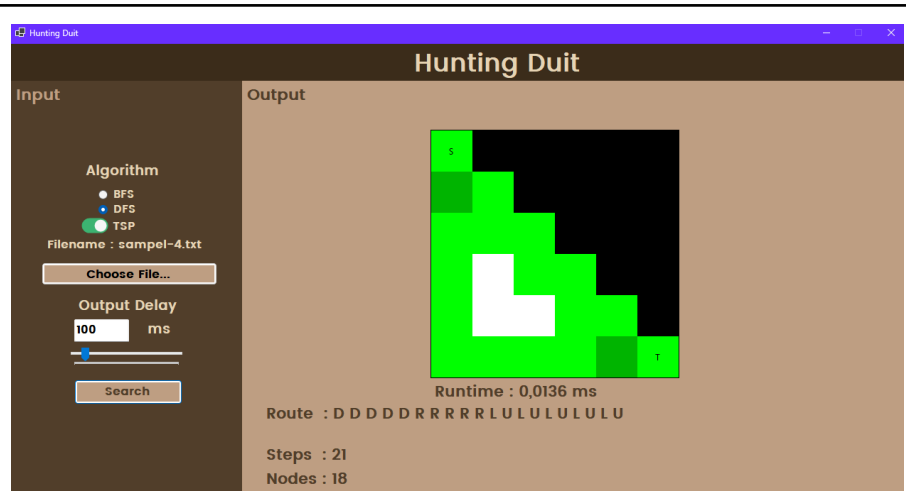
TSP BFS



DFS



TSP DFS



4.5 Analisis Algoritma

Terdapat beberapa pengujian yang dilakukan terhadap program dengan file yang berbeda. Algoritma BFS yang kami gunakan menggunakan metode iteratif. Algoritma akan mengecek apakah tetangga dari node yang sedang dikunjungi apakah memiliki treasure dan belum dikunjungi. Program hanya akan mencari path dimana terdapat treasure. Algoritma DFS yang kami gunakan menggunakan metode *recursive*. Algoritma akan mengecek semua tetangga yang belum pernah dikunjungi dan memasukkannya ke dalam *stack*. Dapat dilihat dari tabel sebelumnya, bahwa metode BFS kami lebih optimal secara frekuensi. Hal ini disebabkan oleh kode program kami dimana priority DFS adalah menuju node yang indeks nya lebih besar terlebih dahulu sehingga bila treasure terletak pada indeks awal, dapat dikatakan metode BFS lebih optimal dari metode DFS. Terlihat pada test ke-3, jika treasure berada pada maze secara mendalam, hasil BFS dan DFS akan sama. Pada test ke-4, dapat dilihat node yang dicek DFS lebih sedikit dari BFS karena treasure berada pada maze bagian dalam. Pada test lainnya, treasure menyebar sehingga metode BFS dapat dikatakan lebih efektif.

BAB 5

Kesimpulan dan Saran

5.1 Kesimpulan

Pada tugas besar 2 IF2211 Strategi Algoritma Semester 2 Tahun Ajaran 2022/2023 ini, kami diminta untuk membuat program berbentuk aplikasi desktop untuk menyelesaikan persoalan maze treasure hunt dan kami telah berhasil membuat aplikasi desktop tersebut. Kami membuat program menggunakan bahasa pemrograman C# dan .NET *framework* untuk mengembangkan aplikasi desktop. Program dapat melakukan pencarian *treasure* dalam maze menggunakan metode BFS dan DFS. Program kami juga dapat melakukan pencarian secara *travelling salesman problem* (TSP). Ketika semua *treasure* sudah ditemukan, maka program akan berhenti mengeksekusi serta menampilkan hasil eksekusi pada aplikasi.

5.2 Saran dan Refleksi

Dalam pengerjaan tugas besar ini, terdapat beberapa kendala seperti penggunaan bahasa pemrograman C# yang belum pernah diajarkan dalam kelas dan pembuatan GUI dengan C#. Namun, kelompok kami berusaha untuk melakukan eksplorasi dari banyak sumber dan mendalaminya sehingga akhirnya dapat menyelesaikan tugas ini dengan lancar.

Daftar Pustaka

<http://csharp.net-informations.com/datagridview/csharp-datagridview-tutorial.htm>

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

<https://stackoverflow.com/questions/10018957/how-to-remove-item-from-list-in-c>

Link to Repository

https://github.com/IceTeaXXD/Tubes2_HuntingDuit

Link to Figma

<https://www.figma.com/file/XAMf5fx6ufNmCOPRmidzt5/hunting-duit-team-library?node-id=0%3A1&t=ZzbtDqD3g4OGsxaZ-1>

Link to Youtube

<https://youtu.be/6hV3yZ2k77A>