

LAPORAN TUGAS BESAR

Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan

Maze Treasure Hunt

IF2211 Strategi Algoritma

Kelompok Yasin

Dosen : Dr. Ir. Rinaldi, M.T.



Anggota Kelompok :

Muhammad Hanan (13521041)

Mutawally Nawwar (13521065)

Bagas Aryo Seto (13521081)

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Bandung

2023

Daftar Isi

Daftar Isi	1
BAB I	2
BAB II	5
2.1. Dasar Teori	5
2.2 C# Desktop Application Development	8
BAB III	9
3.1. Langkah-langkah Pemecahan Masalah	9
3.2. Elemen-elemen Algoritma BFS dan DFS	9
3.3. Ilustrasi Kasus Lain	9
BAB IV	10
4.1. Implementasi Program	10
4.2. Struktur Data dan Spesifikasi Program	13
4.3. Tata Cara Penggunaan Program	13
4.4. Hasil Pengujian	13
4.5. Analisis Desain Solusi Algoritma BFS dan DFS	15
BAB V	16
5.1. Kesimpulan	16
5.2. Saran	16
5.3. Refleksi	16
5.4. Tanggapan Terkait Tugas Besar	16
Daftar Pustaka	17
Lampiran	18

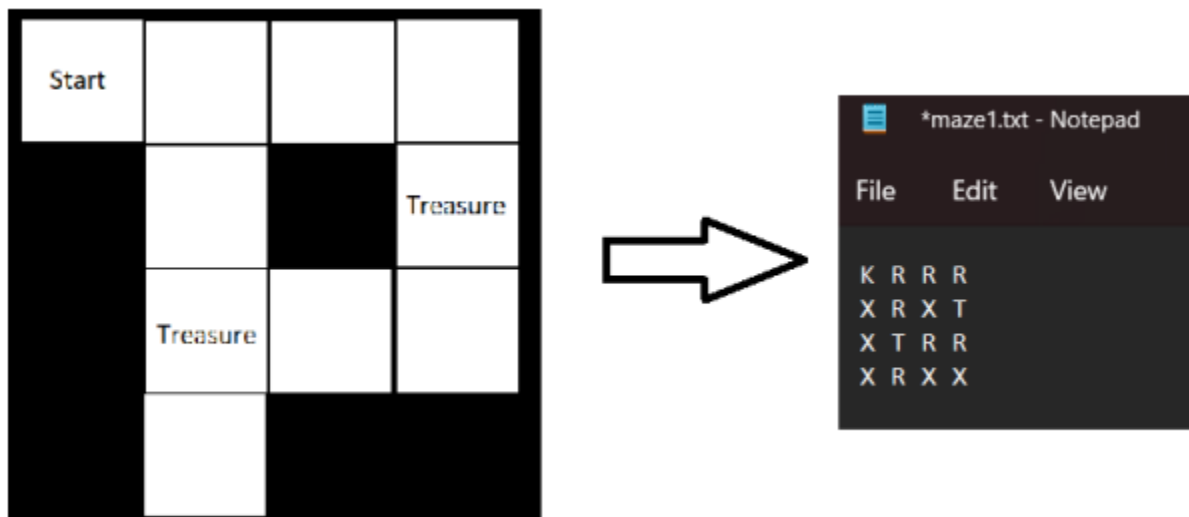
BAB I

DESKRIPSI TUGAS

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

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

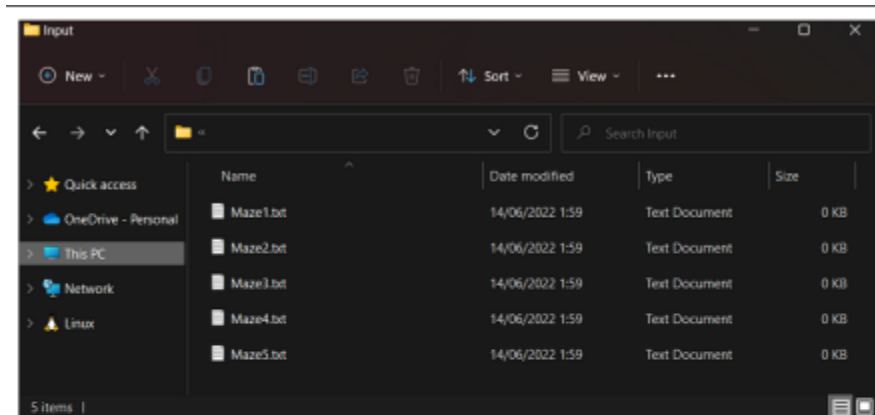
Contoh input:



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:



Input filename

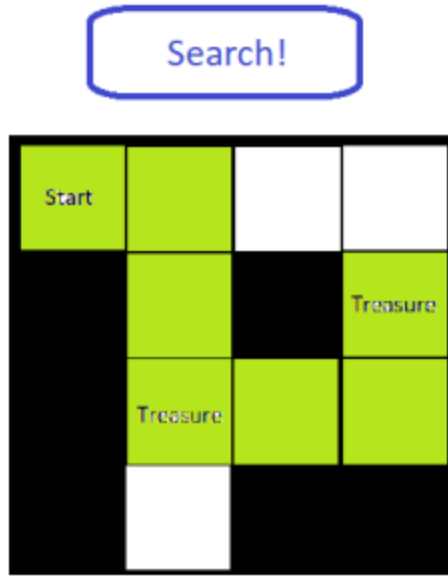
Maze1.txt

Search

Input Nama File

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:



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

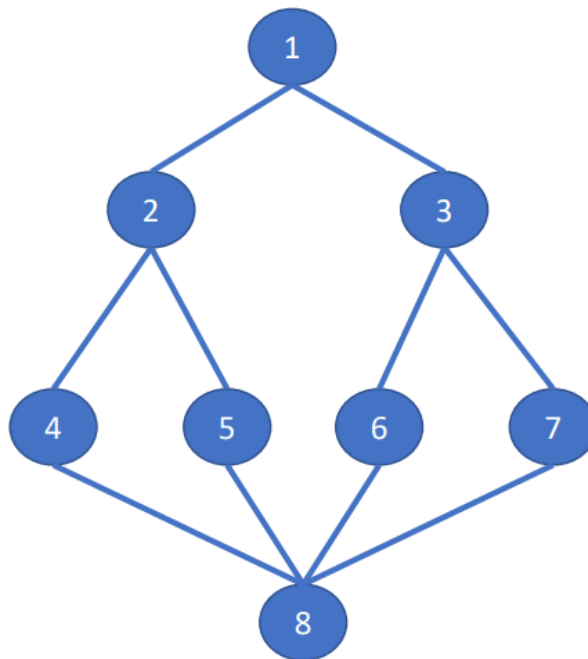
LANDASAN TEORI

2.1. Dasar Teori

Traversal Graf merupakan algoritma yang digunakan untuk mengunjungi seluruh simpul dalam suatu graf secara sistematis. Pada umumnya, traversal digunakan untuk mencari atau memproses data dari suatu graf. Terdapat dua algoritma untuk traversal graf.

1. BFS (*breadth first search*)

Pencarian ini merupakan metode yang melakukan pencarian secara horizontal terlebih dahulu. Dengan kata lain, mencari setiap pada simpul yang sama levelnya, lalu dilanjutkan mencari pada level selanjutnya. Metode ini dapat diimplementasi menggunakan *queue*.



Gambar 2.1 Graf

Pada gambar diatas, urutan simpul-simpul yang dikunjungi dengan menggunakan algoritma BFS adalah 1, 2, 3, 4, 5, 6, 7, 8. Pada prosedur breadth-first search, dapat digunakan adjacency lists dalam merepresentasikan graf $G = (V, E)$. Selain itu, untuk mengetahui simpul yang akan diperiksa, akan digunakan struktur data Queue. Terakhir, untuk mengetahui suatu simpul telah diperiksa atau tidak, akan digunakan struktur data

array atau hash table yang bertipe boolean. Berikut adalah pseudocode umum untuk prosedur breadth-first search.

```

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

  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke layar
}
Deklarasi
w : integer
q : antrian;

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

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

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

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

Algoritma:
  BuatAntrian(q)          { buat antrian kosong }

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

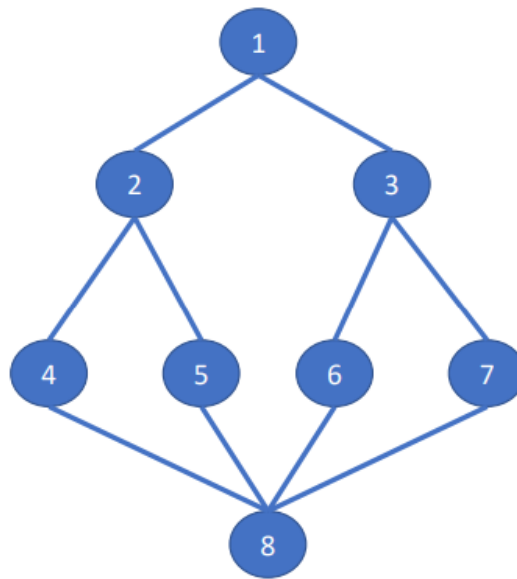
  { kunjungi semua simpul graf selama antrian belum kosong }
  while not AntrianKosong(q) do
    HapusAntrian(q,v)     { simpul v telah dikunjungi, hapus dari
                           antrian }
    for tiap simpul w yang bertetangga dengan simpul v do
      if not dikunjungi[w] then
        write(w)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

```

Gambar 2.2 Algoritma BFS

2. DFS (*depth first search*)

Pencarian ini merupakan metode dimana mencari pada satu simpul dilanjutkan anaknya sampai habis, lalu dilanjutkan pindah ke simpul lainnya. Dengan kata lain akan menuju kedalaman terlebih dahulu kemudian dilanjut ke simpul lain. Metode ini dapat diimplementasi menggunakan *stack* atau *list*.



Gambar 2.3 Graf

Pada gambar diatas, urutan simpul-simpul yang dikunjungi dengan menggunakan algoritma DFS adalah 1, 2, 4, 8, 5, 6, 3, 7. Pada prosedur depth-first search, representasi graf dan struktur data hampir serupa seperti yang digunakan pada prosedur breadth-first search. Akan tetapi, pada depth-first search, tidak akan digunakan struktur data queue, melainkan lebih mirip dengan struktur data stack. Berikut adalah pseudocode dari prosedur depth-first search.

```

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
  
```

Gambar 2.4 Algoritma DFS

2.2 C# Desktop Application Development

C# Desktop Application Development adalah proses pengembangan aplikasi desktop menggunakan bahasa pemrograman C# dan framework .NET. Dalam proses ini, pengembang menggunakan Visual Studio, sebuah Integrated Development Environment (IDE) yang menyediakan berbagai macam fitur untuk memudahkan proses pengembangan, seperti code editor, debugger, dan designer untuk antarmuka pengguna (user interface).

Dalam C# Desktop Application Development, pengembang dapat memanfaatkan berbagai macam teknologi .NET, seperti Windows Presentation Foundation (WPF), Windows Forms, atau Universal Windows Platform (UWP), untuk membuat antarmuka pengguna yang menarik dan responsif. Selain itu, pengembang juga dapat memanfaatkan berbagai macam komponen dan library yang tersedia dalam .NET, seperti ADO.NET untuk mengakses basis data, atau LINQ untuk melakukan query data.

Proses pengembangan aplikasi desktop dengan C# meliputi beberapa tahap, seperti analisis kebutuhan, desain antarmuka pengguna, pemrograman, pengujian, dan distribusi. Selama proses pengembangan, pengembang juga harus memperhatikan masalah keamanan dan performa aplikasi, serta mengoptimalkan kode agar aplikasi dapat berjalan dengan baik di berbagai macam sistem operasi.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-langkah Pemecahan Masalah

Langkah-langkah pemecahan masalah untuk menyelesaikan permasalahan Maze Treasure Hunt ini adalah pertama program akan menerima inputan berupa file txt yang dipilih oleh pengguna untuk dicari penyelesaiannya. File tadi akan dilakukan validasi seperti memastikan inputan hanya mengandung karakter K, R, T, dan X. kemudian memastikan bahwa banyaknya K hanyalah sebanyak 1.

Setelah dilakukan validasi, barulah map divisualisasikan ke layar. Kemudian program akan berjalan sesuai dengan algoritma yang dipilih oleh pengguna, BFS atau DFS, dan apakah ingin mencari jalan pulang kembali ke Krusty Krab dengan algoritma TSP (Travelling salesman problem). Kemudian solusi jalannya akan divisualisasikan pada program.

3.2. Elemen-elemen Algoritma BFS dan DFS

Berdasarkan yang telah dijelaskan sebelumnya, pengaplikasian algoritma BFS menggunakan struktur data *queue* sedangkan algoritma DFS menggunakan struktur data *list* dalam pengaplikasiannya. Berikut merupakan rincian dari alur BFS dan DFS pada program:

1. BFS

Algoritma akan mengambil titik awal sebagai parameter lalu meng-enqueue titik tersebut ke dalam queue terlebih dahulu. Selanjutnya akan dilakukan perulangan selama treasure belum ditemukan atau queue belum kosong. Pertama lakukan dequeue pada queue. Jika cell dequeue bukan merupakan treasure, cell sekitarnya akan dicek dengan prioritas pengecekan kiri, bawah, kanan, atas (L-D-R-U) kemudian cell sekitar akan di enqueue jika cell tersebut valid dan bukan merupakan dinding.

2. DFS

Algoritma akan mengambil titik awal sebagai parameter lalu memasukkan titik tersebut ke dalam list. Selanjutnya akan dilakukan perulangan selama treasure belum ditemukan. Cek setiap cell di sekitar titik awal dengan prioritas pengecekan kiri, bawah, kanan, atas (L-D-R-U) kemudian cell yang valid dan bukan merupakan dinding akan dimasukkan ke dalam list lalu dilakukan pemanggilan rekursif fungsi DFS dengan cell tersebut sebagai

titik awal yang baru. Jika treasure tidak ditemukan pada suatu rute hingga titik terjauh, maka cell akan di-pop atau dihapus dari list.

3.3. Ilustrasi Kasus Lain

Kasus lain yang tidak sama dengan spesifikasi tugas besar adalah pencarian *shortest-path*. Pada kasus ini, dapat diselesaikan menggunakan BFS dan DFS juga, tetapi pada kasus ini perlu dicek pada akhir DFS untuk memilih jalur terpendek yang perlu dilalui. Hal ini perlu dilakukan karena DFS merupakan algoritma penjelajah graf menuju kedalaman terlebih dahulu sehingga tidak secara langsung dapat menentukan jalur terpendek. Sedangkan BFS akan selalu menemukan jalur terpendek walaupun memerlukan ruang yang cukup besar.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Program

1. solveBFS()

```
1 public void solveBFS()
2 {
3     state.current = state.start;
4     state.visitBefore = new bool[state.rows, state.cols];
5     state.path.Add(state.start);
6     state.visitBefore[state.start.Item1, state.start.Item2] = true;
7     while (state.TFounds != state.TCounts)
8     {
9         state.visited = new bool[state.rows, state.cols];
10        BFS(state.current.Item1, state.current.Item2, 'T');
11    }
12 }
```

2. BFS()

```
1 public void BFS(int x, int y, char target)
2 {
3     // make BFS queue then enqueue (x,y)
4     Queue<Tuple<int, int>> queue = new Queue<Tuple<int, int>>();
5     queue.Enqueue(Tuple.Create(x, y));
6
7     // make a matrix of tuple as parent of each points
8     state.parent = new Tuple<int, int>[state.rows, state.cols];
9     state.parent[x, y] = Tuple.Create(-1, -1);
10
11
12     // loop while queue not empty
13     while (queue.Count > 0) {
14         // dequeue as temp then mark as visited
15         Tuple<int, int> temp = queue.Dequeue();
16         state.visited[temp.Item1, temp.Item2] = true;
17
18         state.traversalPath.Add(temp);
19         // Treasure encountered
20         if (state.map[temp.Item1, temp.Item2] == target) {
21             // inc Counter, change current position, change map value to R
22             state.TFounds++;
23             state.current = Tuple.Create(temp.Item1, temp.Item2);
24             state.map[temp.Item1, temp.Item2] = 'R';
25
26             // List path from (x,y) to T then concat to path
27             List<Tuple<int, int>> tempPath = new List<Tuple<int, int>>();
28             while (!temp.Equals(new Tuple<int, int>(x, y))) {
29                 tempPath.Add(temp);
30                 temp = state.parent[temp.Item1, temp.Item2];
31                 state.visitBefore[temp.Item1, temp.Item2] = true;
32             }
33             tempPath.Reverse();
34             state.path.AddRange(tempPath);
35
36             break;
37         }
38     }
```

```

1 // enqueue all possible road
2 var possibleDir = new List<Tuple<int, int>>();
3 for (int i = 3; i >= 0; i--) {
4     int x2 = temp.Item1 + state.dx[i];
5     int y2 = temp.Item2 + state.dy[i];
6
7     if (goCheck(x2, y2))
8     {
9         state.visited[x2, y2] = true;
10
11         if (state.visitBefore[x2, y2])
12         {
13             possibleDir.Add(Tuple.Create(x2, y2));
14         }
15         else
16         {
17             possibleDir.Insert(0, Tuple.Create(x2, y2));
18         }
19     }
20 }
21
22 foreach (var dir in possibleDir)
23 {
24     int xd = dir.Item1, yd = dir.Item2;
25     state.parent[xd, yd] = temp;
26     queue.Enqueue(dir);
27 }
28 }
29 }

```

3. solveDFS()

```

1 public void solveDFS()
2 {
3     state.current = state.start;
4     state.visitBefore = new bool[state.rows, state.cols];
5     while (state.TFounds != state.TCounts)
6     {
7         state.anotherWay = false;
8         state.visited = new bool[state.rows, state.cols];
9         DFS(state.current.Item1, state.current.Item2, 'T');
10    }
11 }

```

4. DFS()

```

1 public bool DFS(int x, int y, char target)
2 {
3     // mark as visited
4     state.visited[x, y] = true;
5     state.visitBefore[x, y] = true;
6     Tuple<int, int> temp = Tuple.Create(x, y);
7     state.traversalPath.Add(temp);
8
9     // add (x,y) to path
10    if (state.path.Count == 0 || !state.path.Last().Equals(temp))
11    {
12        state.path.Add(temp);
13    }
14
15    // collect T if (x,y) is a treasure
16    if (state.map[x, y] == target)
17    {
18        // change T value to R, inc TFounds, change current position
19        state.map[x, y] = 'R';
20        state.TFounds++;
21        state.current = Tuple.Create(x, y);
22
23        // return true
24        return true;
25    }

```

```

1 // recursive
2 var possibleDir = new List<Tuple<int, int>>();
3 int cnt = 0;
4 for (int i = 0; i < 4; i++)
5 {
6     int x2 = x + state.dx[i];
7     int y2 = y + state.dy[i];
8     if (goCheck(x2, y2))
9     {
10         if (!state.visitBefore[x2, y2])
11         {
12             cnt++;
13         }
14         state.anotherWay = state.anotherWay || !state.visitBefore[x2, y2];
15         possibleDir.Add(Tuple.Create(x2, y2));
16     }
17 }
18
19 foreach (var dir in possibleDir)
20 {
21     int x2 = dir.Item1, y2 = dir.Item2;
22     if ((!state.anotherWay && state.visitBefore[x2, y2])
23         || !state.visitBefore[x2, y2] || cnt == 0)
24     {
25         if (DFS(x2, y2, target))
26         {
27             return true;
28         }
29     }
30 }
31
32 // remove point from path
33 state.path.RemoveAt(state.path.Count - 1);
34 state.visited[x, y] = false;
35 return false;
36 }

```

4.2. Struktur Data dan Spesifikasi Program

Berikut merupakan struktur data dan spesifikasi program yang kelompok kami gunakan pada tugas kali ini :

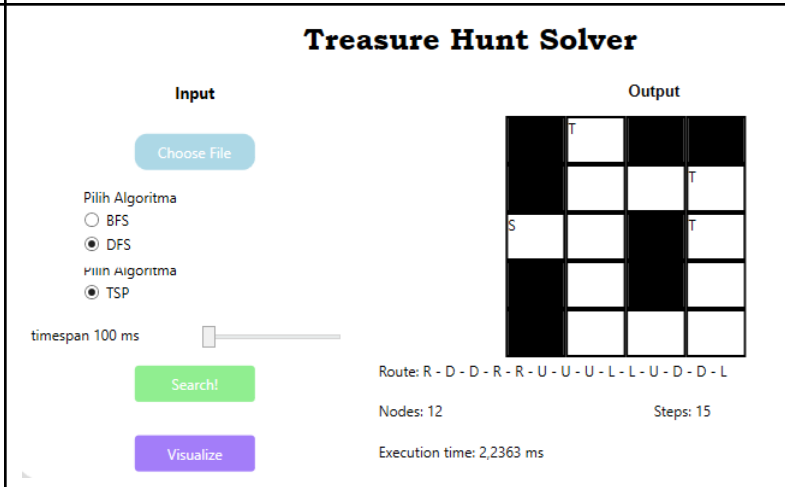
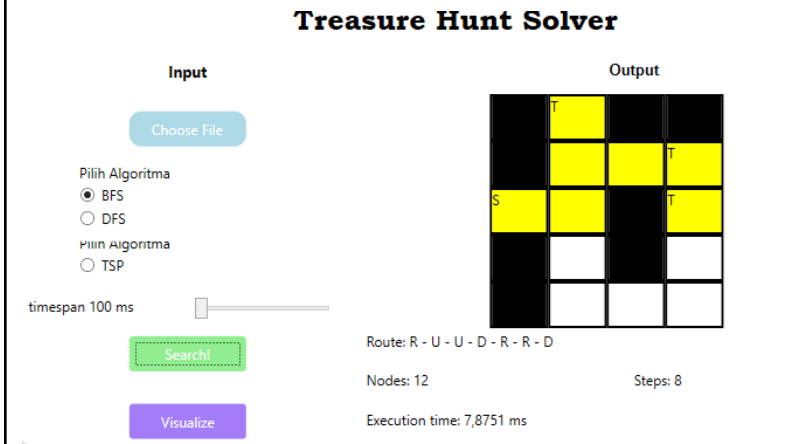
1. Queue, dalam mengimplementasikan struktur data queue pada program kami, kami menggunakan *collection* pada kelas `System.Collections.Generic`. Queue ini digunakan untuk menampung node yang merupakan kotak yang dapat dilalui pada maze yang nantinya akan dikunjungi saat melakukan transversal dengan menggunakan metode BFS (*Breadth First Search*). Berikut implementasi method yang digunakan:
 - a. Enqueue, berfungsi untuk memasukkan node ke dalam antrian queue menggunakan aturan First In First Out (FIFO).
 - b. Dequeue, berfungsi untuk mengeluarkan node dari dalam antrian queue menggunakan aturan First In First Out (FIFO).
2. Stack, sama seperti Queue, dalam pengimplementasian Stack dalam program kami, kami juga menggunakan *collection* pada kelas `System.Collections.Generic`. Stack ini digunakan untuk menyimpan tumpukan node yang merupakan kotak yang dapat dilalui pada maze yang nantinya akan dikunjungi saat melakukan transversal dengan menggunakan metode DFS (*Depth First Search*). Berikut implementasi method yang digunakan:
 - a. Push, berfungsi untuk memasukkan sebuah node ke dalam tumpukan node menggunakan aturan Last In First Out (LIFO).
 - b. Pull, berfungsi untuk mengeluarkan sebuah node dari dalam tumpukan node menggunakan aturan Last In First Out (LIFO).
3. Point, point digunakan sebagai dasar dari sebuah maze yang digunakan untuk menentukan posisi-posisi dari suatu tile. Point ini berisi koordinat kartesian yang memuat integer X dan Y.
4. List, list adalah salah satu jenis tipe data yang umum digunakan dalam pemrograman. Dalam bahasa pemrograman, list biasanya digunakan untuk menyimpan kumpulan data atau objek. Pada program ini, list digunakan untuk menampung tuple-tuple berupa point pada maze yang digunakan.

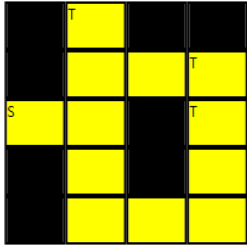
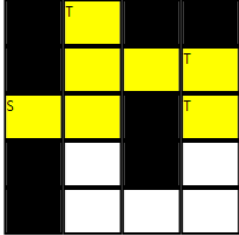
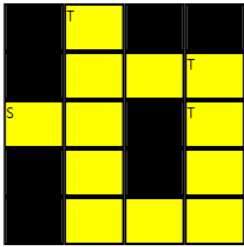
4.3. Tata Cara Penggunaan Program

1. Clone repository atau download sebagai file ZIP lalu ekstrak
2. Buka file executable `bin/Tubes2_Yasin.exe`
3. Pilih file sebagai input
4. Pilih algoritma yang akan digunakan
5. Klik tombol search
6. Klik visualize

4.4. Hasil Pengujian

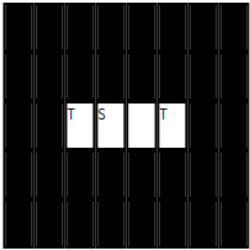
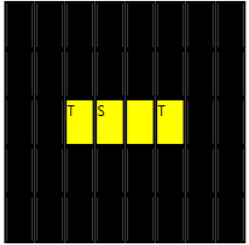
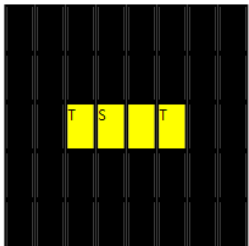
1. sampel-1.txt

Penjelasan	Tampilan
Tampilan awal program	 <p>Treasure Hunt Solver</p> <p>Input</p> <p>Choose File</p> <p>Pilih Algoritma</p> <p><input type="radio"/> BFS</p> <p><input checked="" type="radio"/> DFS</p> <p>Pilih Algoritma</p> <p><input checked="" type="radio"/> TSP</p> <p>timespan 100 ms</p> <p>Search!</p> <p>Visualize</p> <p>Output</p> <p>Route: R - D - D - R - R - U - U - U - L - L - U - D - D - L</p> <p>Nodes: 12</p> <p>Steps: 15</p> <p>Execution time: 2,2363 ms</p>
Pencarian solusi dengan BFS	 <p>Treasure Hunt Solver</p> <p>Input</p> <p>Choose File</p> <p>Pilih Algoritma</p> <p><input checked="" type="radio"/> BFS</p> <p><input type="radio"/> DFS</p> <p>Pilih Algoritma</p> <p><input type="radio"/> TSP</p> <p>timespan 100 ms</p> <p>Search!</p> <p>Visualize</p> <p>Output</p> <p>Route: R - U - U - D - R - R - D</p> <p>Nodes: 12</p> <p>Steps: 8</p> <p>Execution time: 7,8751 ms</p>

<p>Pencarian solusi dengan DFS</p>	<div data-bbox="948 210 1287 241"> <h3>Treasure Hunt Solver</h3> </div> <div data-bbox="673 268 1396 657"> <div> <div>Input</div> <div>Choose File</div> <div> Pilih Algoritma <input type="radio"/> BFS <input checked="" type="radio"/> DFS Pilih Algoritma <input type="radio"/> TSP </div> <div> timespan 100 ms <div></div> </div> <div>Search!</div> <div>Visualize</div> </div> <div> <div>Output</div> <div>  </div> <div> Route: R - D - D - R - R - U - U - L - L - U Nodes: 12 Execution time: 1,7435 ms </div> <div> Steps: 12 </div> </div> </div>
<p>Pencarian solusi dengan BFS dan kembali ke tempat semula dengan TSP</p>	<div data-bbox="941 707 1268 739"> <h3>Treasure Hunt Solver</h3> </div> <div data-bbox="673 764 1373 1136"> <div> <div>Input</div> <div>Choose File</div> <div> Pilih Algoritma <input checked="" type="radio"/> BFS <input type="radio"/> DFS Pilih Algoritma <input checked="" type="radio"/> TSP </div> <div> timespan 100 ms <div></div> </div> <div>Search!</div> <div>Visualize</div> </div> <div> <div>Output</div> <div>  </div> <div> Route: R - U - U - D - R - R - D - U - L - L - D - L Nodes: 12 Execution time: 2,2707 ms </div> <div> Steps: 13 </div> </div> </div>
<p>Pencarian solusi dengan DFS dan kembali ke tempat semula dengan TSP</p>	<div data-bbox="948 1186 1284 1218"> <h3>Treasure Hunt Solver</h3> </div> <div data-bbox="673 1243 1390 1629"> <div> <div>Input</div> <div>Choose File</div> <div> Pilih Algoritma <input type="radio"/> BFS <input checked="" type="radio"/> DFS Pilih Algoritma <input checked="" type="radio"/> TSP </div> <div> timespan 100 ms <div></div> </div> <div>Search!</div> <div>Visualize</div> </div> <div> <div>Output</div> <div>  </div> <div> Route: R - D - D - R - R - U - U - L - L - U - D - D - L Nodes: 12 Execution time: 2,2363 ms </div> <div> Steps: 15 </div> </div> </div>

2. sampel-2.txt

<p>Penjelasan</p>	<p>Tampilan</p>
-------------------	-----------------

Tampilan awal program	<div data-bbox="954 210 1295 243"> <h3>Treasure Hunt Solver</h3> </div> <div data-bbox="821 268 867 294"> <p>Input</p> </div> <div data-bbox="805 331 883 352"> <p>Choose File</p> </div> <div data-bbox="727 378 824 399"> <p>Pilih Algoritma</p> </div> <div data-bbox="727 403 776 424"> <p><input type="radio"/> BFS</p> </div> <div data-bbox="727 428 776 449"> <p><input checked="" type="radio"/> DFS</p> </div> <div data-bbox="727 453 824 474"> <p>Pilih Algoritma</p> </div> <div data-bbox="727 478 776 499"> <p><input checked="" type="radio"/> TSP</p> </div> <div data-bbox="675 516 789 537"> <p>timespan 100 ms</p> </div> <div data-bbox="821 562 867 583"> <p>Search!</p> </div> <div data-bbox="812 634 876 655"> <p>Visualize</p> </div> <div data-bbox="1282 268 1344 294"> <p>Output</p> </div> <div data-bbox="1159 302 1406 550">  </div> <div data-bbox="1029 554 1386 575"> <p>Route: R - D - D - R - R - U - U - U - L - L - U - D - D - L</p> </div> <div data-bbox="1029 592 1094 613"> <p>Nodes: 12</p> </div> <div data-bbox="1308 592 1370 613"> <p>Steps: 15</p> </div> <div data-bbox="1029 634 1198 655"> <p>Execution time: 2,2363 ms</p> </div>
Pencarian solusi dengan BFS	<div data-bbox="945 711 1286 745"> <h3>Treasure Hunt Solver</h3> </div> <div data-bbox="818 770 863 795"> <p>Input</p> </div> <div data-bbox="802 833 880 854"> <p>Choose File</p> </div> <div data-bbox="724 879 821 900"> <p>Pilih Algoritma</p> </div> <div data-bbox="724 905 773 926"> <p><input checked="" type="radio"/> BFS</p> </div> <div data-bbox="724 930 773 951"> <p><input type="radio"/> DFS</p> </div> <div data-bbox="724 955 821 976"> <p>Pilih Algoritma</p> </div> <div data-bbox="724 980 773 1001"> <p><input type="radio"/> TSP</p> </div> <div data-bbox="672 1018 786 1039"> <p>timespan 100 ms</p> </div> <div data-bbox="818 1064 863 1085"> <p>Search!</p> </div> <div data-bbox="808 1136 873 1157"> <p>Visualize</p> </div> <div data-bbox="1269 770 1331 795"> <p>Output</p> </div> <div data-bbox="1146 804 1393 1047">  </div> <div data-bbox="1019 1052 1143 1073"> <p>Route: L - R - R - R</p> </div> <div data-bbox="1019 1089 1084 1110"> <p>Nodes: 4</p> </div> <div data-bbox="1292 1089 1354 1110"> <p>Steps: 5</p> </div> <div data-bbox="1019 1131 1188 1152"> <p>Execution time: 7,9323 ms</p> </div>
Pencarian solusi dengan DFS	<div data-bbox="948 1209 1289 1243"> <h3>Treasure Hunt Solver</h3> </div> <div data-bbox="821 1268 867 1293"> <p>Input</p> </div> <div data-bbox="805 1331 883 1352"> <p>Choose File</p> </div> <div data-bbox="727 1377 824 1398"> <p>Pilih Algoritma</p> </div> <div data-bbox="727 1402 776 1423"> <p><input type="radio"/> BFS</p> </div> <div data-bbox="727 1428 776 1449"> <p><input checked="" type="radio"/> DFS</p> </div> <div data-bbox="727 1453 824 1474"> <p>Pilih Algoritma</p> </div> <div data-bbox="727 1478 776 1499"> <p><input type="radio"/> TSP</p> </div> <div data-bbox="675 1516 789 1537"> <p>timespan 100 ms</p> </div> <div data-bbox="821 1562 867 1583"> <p>Search!</p> </div> <div data-bbox="812 1633 876 1654"> <p>Visualize</p> </div> <div data-bbox="1276 1268 1338 1293"> <p>Output</p> </div> <div data-bbox="1153 1302 1399 1545">  </div> <div data-bbox="1023 1549 1149 1570"> <p>Route: L - R - R - R</p> </div> <div data-bbox="1023 1587 1088 1608"> <p>Nodes: 4</p> </div> <div data-bbox="1299 1587 1360 1608"> <p>Steps: 5</p> </div> <div data-bbox="1023 1629 1198 1650"> <p>Execution time: 4,9188 ms</p> </div>

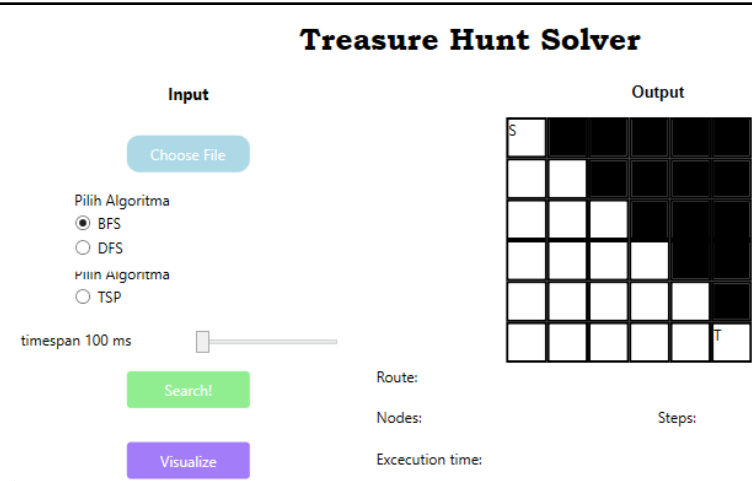
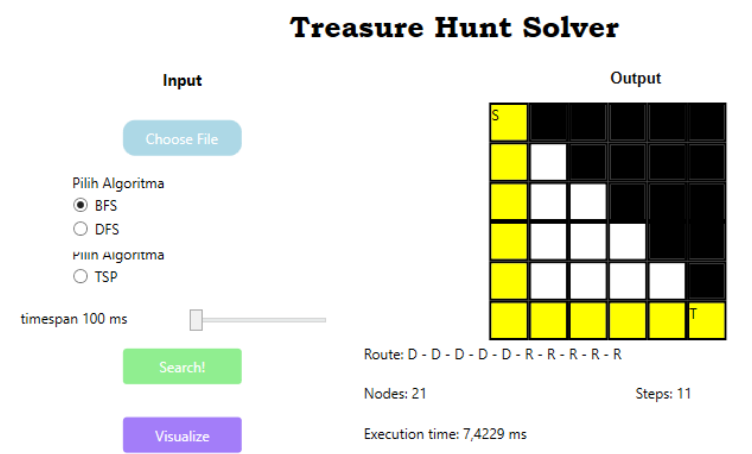
<p>Pencarian solusi dengan BFS dan kembali ke tempat semula dengan TSP</p>	<div data-bbox="950 210 1289 241"> <h3>Treasure Hunt Solver</h3> </div> <div data-bbox="657 262 1396 661"> <div> <div>Input</div> <div> <div>Choose File</div> <div> <div>Pilih Algoritma</div> <div> <input checked="" type="radio"/> BFS <input type="radio"/> DFS </div> <div> <div>Pilih Algoritma</div> <div> <input checked="" type="radio"/> TSP </div> </div> <div> <div>timespan 100 ms</div> <div></div> </div> <div>Search!</div> <div>Visualize</div> </div> <div> <div>Output</div> <div> </div> <div> <div>Route: L - R - R - R - L - L</div> <div>Nodes: 4</div> <div>Steps: 7</div> <div>Execution time: 4,0419 ms</div> </div> </div> </div> </div></div>
<p>Pencarian solusi dengan DFS dan kembali ke tempat semula dengan TSP</p>	<div data-bbox="941 711 1273 743"> <h3>Treasure Hunt Solver</h3> </div> <div data-bbox="657 764 1396 1163"> <div> <div>Input</div> <div> <div>Choose File</div> <div> <div>Pilih Algoritma</div> <div> <input type="radio"/> BFS <input checked="" type="radio"/> DFS </div> <div> <div>Pilih Algoritma</div> <div> <input checked="" type="radio"/> TSP </div> </div> <div> <div>timespan 100 ms</div> <div></div> </div> <div>Search!</div> <div>Visualize</div> </div> <div> <div>Output</div> <div> </div> <div> <div>Route: L - R - R - R - L - L</div> <div>Nodes: 4</div> <div>Steps: 7</div> <div>Execution time: 5,1504 ms</div> </div> </div> </div> </div></div>

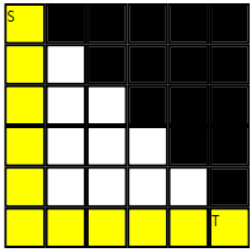
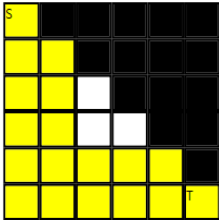
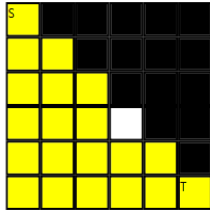
3. sampel-3.txt

Penjelasan	Tampilan
<p>Tampilan awal program</p>	<div data-bbox="966 1350 1325 1381"> <h3>Treasure Hunt Solver</h3> </div> <div data-bbox="657 1402 1396 1822"> <div> <div>Input</div> <div> <div>Choose File</div> <div> <div>Pilih Algoritma</div> <div> <input checked="" type="radio"/> BFS <input type="radio"/> DFS </div> <div> <div>Pilih Algoritma</div> <div> <input type="radio"/> TSP </div> </div> <div> <div>timespan 100 ms</div> <div></div> </div> <div>Search!</div> <div>Visualize</div> </div> <div> <div>Output</div> <div> <div>Pastikan file hanya berisi karakter T,R,X,K</div> <div> <div>Route:</div> <div>Nodes:</div> <div>Execution time:</div> </div> <div>Steps:</div> </div> </div> </div> </div></div>

Pencarian solusi dengan BFS	-
Pencarian solusi dengan DFS	-
Pencarian solusi dengan BFS dan kembali ke tempat semula dengan TSP	-
Pencarian solusi dengan DFS dan kembali ke tempat semula dengan TSP	-

4. sampel-4.txt

Penjelasan	Tampilan
Tampilan awal program	
Pencarian solusi dengan BFS	

<p>Pencarian solusi dengan DFS</p>	<div data-bbox="948 212 1287 243"> Treasure Hunt Solver </div> <div data-bbox="670 268 1419 659"> <div> <div>Input</div> <div>Choose File</div> <div> Pilih Algoritma <input type="radio"/> BFS <input checked="" type="radio"/> DFS Pilih Algoritma <input type="radio"/> TSP </div> <div> timespan 100 ms <div></div> </div> <div>Search!</div> <div>Visualize</div> </div> <div> <div>Output</div>  <div> Route: D - D - D - D - D - R - R - R - R - R Nodes: 21 Execution time: 4,0787 ms </div> <div>Steps: 11</div> </div> </div>
<p>Pencarian solusi dengan BFS dan kembali ke tempat semula dengan TSP</p>	<div data-bbox="915 716 1218 747"> Treasure Hunt Solver </div> <div data-bbox="670 764 1419 1113"> <div> <div>Input</div> <div>Choose File</div> <div> Pilih Algoritma <input checked="" type="radio"/> BFS <input type="radio"/> DFS Pilih Algoritma <input checked="" type="radio"/> TSP </div> <div> timespan 100 ms <div></div> </div> <div>Search!</div> <div>Visualize</div> </div> <div> <div>Output</div>  <div> Route: D - D - D - D - D - R - R - R - R - R - L - U - L - L - U - U - L - U Nodes: 21 Execution time: 3,4117 ms </div> <div>Steps: 21</div> </div> </div>
<p>Pencarian solusi dengan DFS dan kembali ke tempat semula dengan TSP</p>	<div data-bbox="906 1169 1195 1201"> Treasure Hunt Solver </div> <div data-bbox="670 1218 1419 1545"> <div> <div>Input</div> <div>Choose File</div> <div> Pilih Algoritma <input type="radio"/> BFS <input checked="" type="radio"/> DFS Pilih Algoritma <input checked="" type="radio"/> TSP </div> <div> timespan 100 ms <div></div> </div> <div>Search!</div> <div>Visualize</div> </div> <div> <div>Output</div>  <div> Route: D - D - D - D - D - R - R - R - R - R - L - U - L - L - U - R - U - L - U - L - U Nodes: 21 Execution time: 3,2687 ms </div> <div>Steps: 23</div> </div> </div>

5. sampel-5.txt

Penjelasan	Tampilan
------------	----------

Tampilan awal program	<div><div>Treasure Hunt Solver</div><div><div>Input</div><div><div>Choose File</div><div><div>Pilih Algoritma</div><div><div><input checked="" type="radio"/> BFS</div><div><input type="radio"/> DFS</div></div><div><div>Pilih Algoritma</div><div><input type="radio"/> TSP</div></div></div><div><div>timespan 100 ms</div><div></div></div><div><div>Search!</div><div>Visualize</div></div></div><div><div>Output</div><div><table><tr><td>S</td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr></table><div><div>Route:</div><div>Nodes:</div><div>Execution time:</div></div><div>Steps:</div></div></div></div></div>	S	T				T				T				T				T				T				T				T				T				T		
S	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
Pencarian solusi dengan BFS	<div><div>Treasure Hunt Solver</div><div><div>Input</div><div><div>Choose File</div><div><div>Pilih Algoritma</div><div><div><input checked="" type="radio"/> BFS</div><div><input type="radio"/> DFS</div></div><div><div>Pilih Algoritma</div><div><input type="radio"/> TSP</div></div></div><div><div>timespan 100 ms</div><div></div></div><div><div>Search!</div><div>Visualize</div></div></div><div><div>Output</div><div><table><tr><td>S</td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr></table><div><div>Route: R - D - D - D - D - D - D - D - D - D</div><div>Nodes: 23</div><div>Execution time: 9,8795 ms</div></div><div>Steps: 12</div></div></div></div></div>	S	T				T				T				T				T				T				T				T				T				T		
S	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
Pencarian solusi dengan DFS	<div><div>Treasure Hunt Solver</div><div><div>Input</div><div><div>Choose File</div><div><div>Pilih Algoritma</div><div><div><input type="radio"/> BFS</div><div><input checked="" type="radio"/> DFS</div></div><div><div>Pilih Algoritma</div><div><input type="radio"/> TSP</div></div></div><div><div>timespan 100 ms</div><div></div></div><div><div>Search!</div><div>Visualize</div></div></div><div><div>Output</div><div><table><tr><td>S</td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr><tr><td></td><td>T</td><td></td><td></td></tr></table><div><div>Route: R - D - D - D - D - D - D - D - D - D</div><div>Nodes: 23</div><div>Execution time: 5,5611 ms</div></div><div>Steps: 12</div></div></div></div></div>	S	T				T				T				T				T				T				T				T				T				T		
S	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								
	T																																								

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dalam pembuatan Maze Treasure Hunt Finder ini, algoritma BFS dan DFS dapat diimplementasikan menggunakan bahasa pemrograman C#. Melalui mata kuliah IF2211 - Strategi Algoritma, strategi dasar dari kedua algoritma tersebut telah dipelajari. Berdasarkan hasil pembuatan program ini, dapat disimpulkan bahwa penggunaan algoritma BFS dan DFS dapat menjadi solusi efektif untuk menemukan treasure pada Maze Treasure Hunt ini.

5.2. Saran

Saran yang dapat kelompok kami berikan untuk tugas besar kali ini adalah semoga kedepannya asisten dapat memberikan contoh test case sesegera mungkin setelah perlisian tugas besar ini.

5.3. Refleksi

Pembuatan tugas besar untuk mata kuliah IF2211 Strategi Algoritma merupakan pengalaman yang sangat berharga bagi penulis. Selama mengerjakan tugas ini, penulis belajar bagaimana pengimplementasian strategi algoritma dapat membantu memecahkan masalah sehari-hari, seperti dalam pembuatan program untuk mencari treasure pada suatu labirin. Meskipun begitu, terdapat beberapa kekurangan pada program yang dibuat dan penulis berharap agar di masa depan dapat mengembangkan algoritma yang lebih baik dan efektif.

5.4. Tanggapan Terkait Tugas Besar

Tanggapan terkait tugas besar ke 2 mata kuliah Strategi Algoritma ini adalah saat awal awal masih kaget saat menggunakan aplikasi visual studio, mulai dari saat set up, kemudian saat menggunakannya untuk membangun program tugas besar ini, tetapi kami tetap dapat insight yang sangat banyak karena telah mencoba platform baru ini

Daftar Pustaka

1. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>
2. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>
3. <https://www.geeksforgeeks.org/rat-in-a-maze/>

Lampiran

Link Repository Github: https://github.com/bagas003/Tubes2_Yasin

Link Video Youtube: <https://youtu.be/ufJRU8tWlpY>