

LAPORAN TUGAS BESAR 2
IF-2211 STRATEGI ALGORITMA
PENGAPLIKASIAN ALGORITMA BFS DAN DFS DALAM
MENYELESAIKAN PERSOALAN MAZE TREASURE HUNT



Disusun oleh:

Kelompok 43 - Duelist Sejati

Anggota:

Christian Albert Hasiholan	13521078
Haidar Hamda	13521105
Edia Zaki Naufal Ilman	13521141

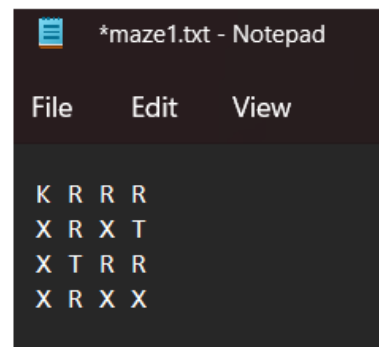
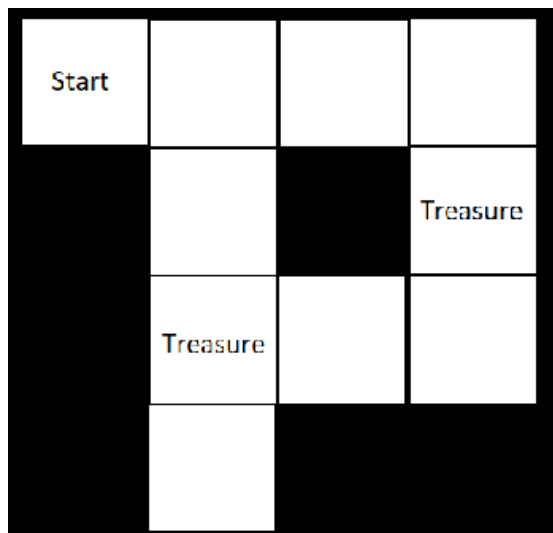
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

BAB I

DESKRIPSI TUGAS

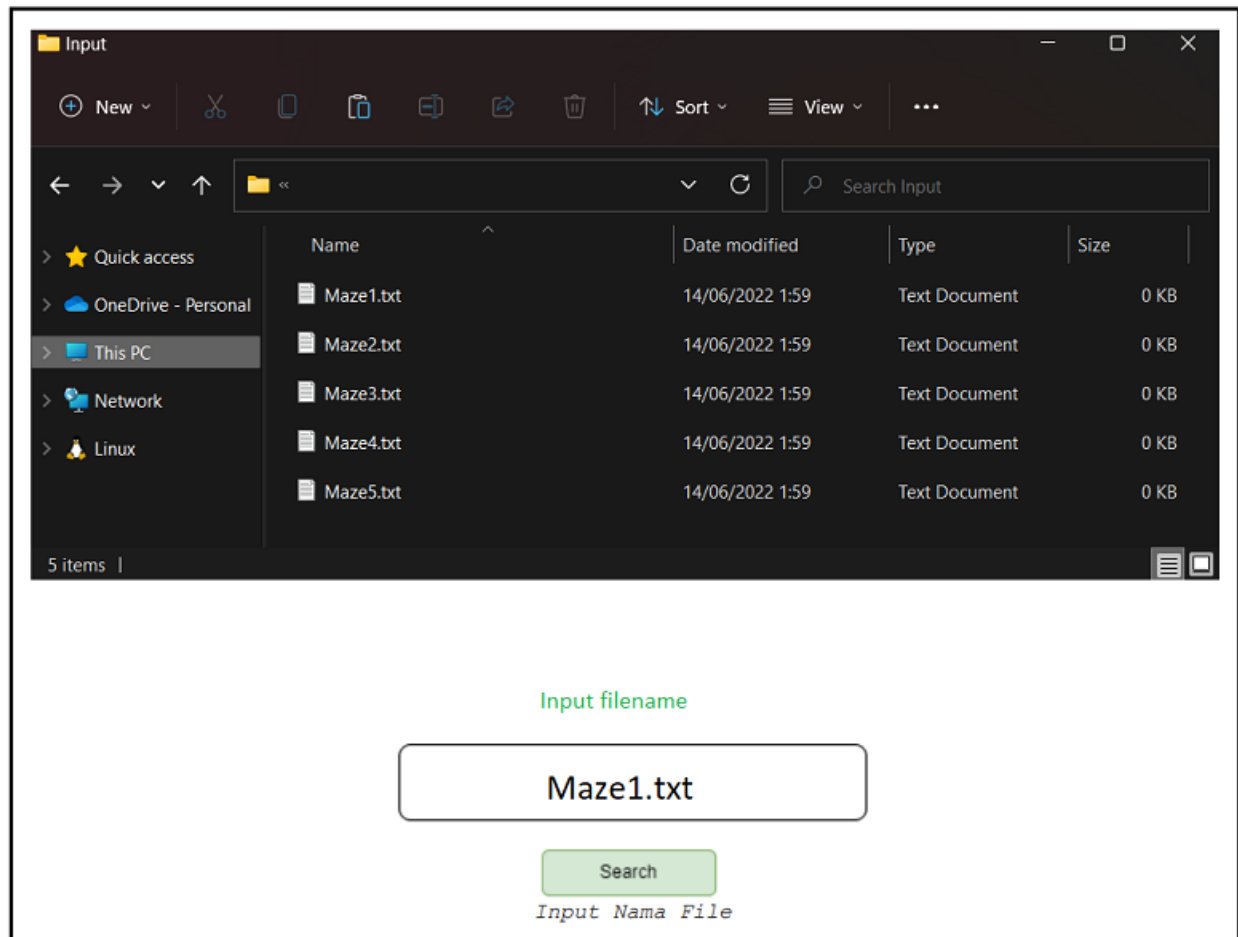
Dalam tugas besar ini, kelompok kami diminta untuk membangun sebuah aplikasi dengan GUI sederhana yang dapat mengimplementasikan BFS dan DFS untuk mendapatkan rute memperoleh seluruh treasure atau harta karun yang ada. Program dapat menerima dan membaca input sebuah file txt yang berisi maze yang akan 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

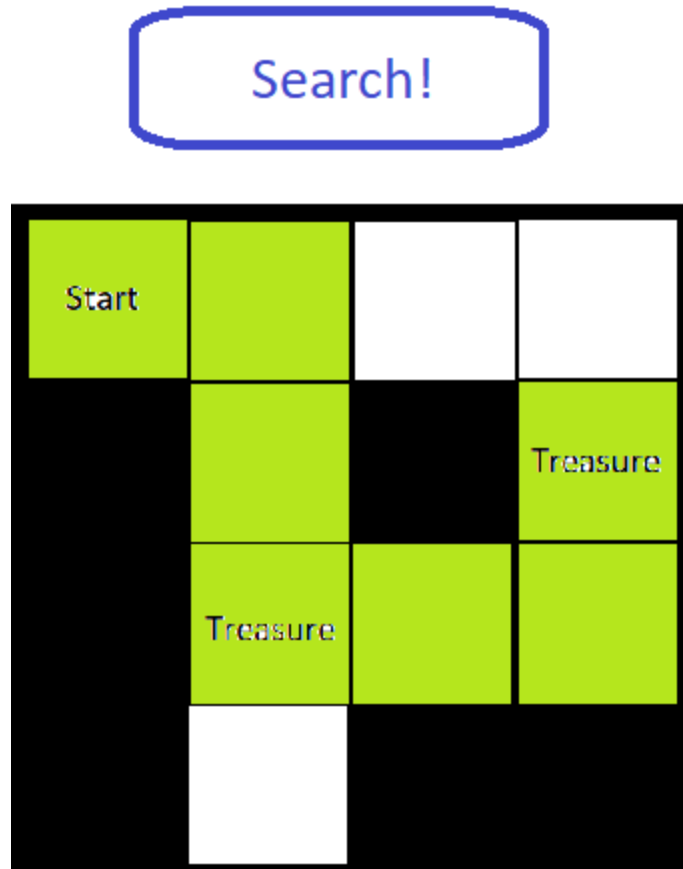


Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), dapat ditelusuri grid (simpul) yang mungkin dikunjungi hingga ditemukan rute solusi, baik secara melebar ataupun mendalam bergantung alternatif algoritma yang dipilih. Rute solusi adalah rute yang memperoleh seluruh treasure pada maze. Rute yang diperoleh dengan algoritma

BFS dan DFS dapat berbeda, dan banyak langkah yang dibutuhkan pun menjadi berbeda. Prioritas arah simpul yang dibangkitkan dibebaskan dan tidak ada pergerakan secara diagonal. Kelompok 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.



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. 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.



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

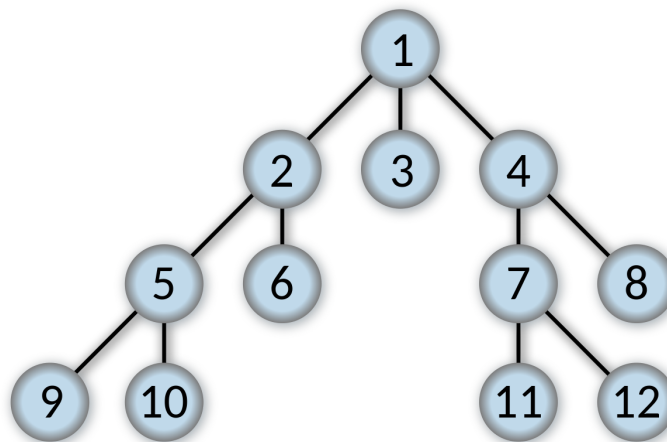
LANDASAN TEORI

1. Traversal Graph

Travesal graph adalah sebuah algoritma yang memiliki tujuan mengunjungi satu atau lebih simpul pada graf dengan cara yang sistematis. Terdapat dua jenis pencarian solusi, yaitu pencarian tanpa informasi (*uninformed/blind search*) dan pencarian dengan informasi (*informed search*). Pada *uninformed search*, algoritma tidak dibekali tambahan informasi apapun, dan beberapa contoh algoritmanya adalah BFS (*Breadth First Search*) dan DFS (*Depth First Search*). Pada *informed search*, algoritma pencarian berbasis heuristik dengan contoh algoritmanya adalah *Best First Search* dan A*.

2. Breadth First Search

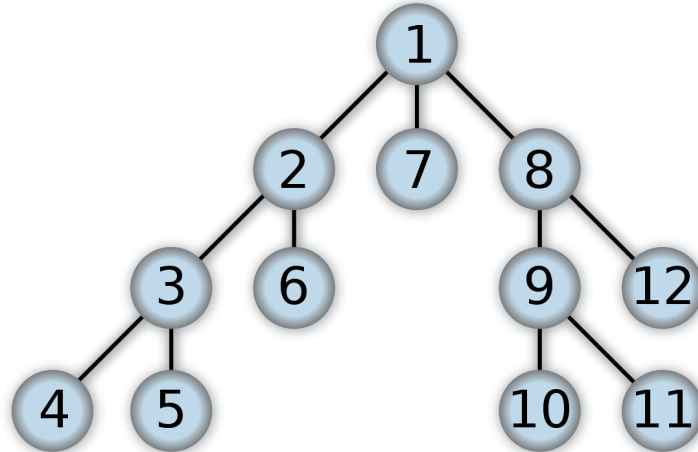
Breadth First Search (BFS) adalah algoritma yang melakukan pencarian secara mendalam. Dalam metoda BFS ini, semua simpul dengan level n dikunjungi terlebih dahulu sebelum mengunjungi simpul-simpul dengan level $n+1$. Berikut ilustrasi langkah pencarian graf dengan BFS:



3. Depth First Search

Depth First Search (DFS) adalah algoritma yang melakukan pencarian secara melebar. Dalam metoda DFS ini, algoritma akan memproses sebuah node kemudian memproses seluruh

children dari node tersebut secara keseluruhan dengan arah vertikal terlebih dahulu sebelum lanjut ke node berikutnya yang setara dengan node awal yang diproses. Berikut ilustrasi langkah pencarian graf dengan DFS:



4. Travelling Salesman Problem

Travelling Salesman Problem (TSP) merupakan sebuah permasalahan yang tujuannya adalah mencari sebuah rute perjalanan dengan jarak terpendek sehingga traversal pada setiap node hanya dilakukan sekali dan kembali ke node awal. Persoala TSP merupakan salah satu contoh kasus menemukan sirkuit hamilton pada graf.

BAB III

APLIKASI BFS DAN DFS

1. Langkah Pemecahan Masalah

Langkah - langkah pemecahan masalah dengan algoritma BFS:

1. Menentukan simpul awal.
2. Membuat antrian kosong yang akan berisi simpul dan rute yang telah dikunjungi.
3. Membuat daftar kosong yang akan berisi simpul yang telah dikunjungi.
4. Selama antrian belum kosong, lakukan dequeue untuk memperoleh simpul yang harus dikunjungi berikutnya. Jika simpul yang sedang dikunjungi adalah “Treasure”, simpan rute yang telah dikunjungi dan break. Tambahkan simpul ke dalam daftar. Tambahkan simpul yang bertetangga dengan simpul yang sedang dikunjungi ke dalam antrian.
5. Ulangi langkah 4 sampai semua “Treasure” ditemukan.

Langkah - langkah pemecahan masalah dengan algoritma DFS:

1. Menentukan simpul awal.
2. Menentukan banyak treasure pada map.
3. Membuat stack kosong yang akan diisi dengan simpul yang akan dilalui dan rute menuju simpul tersebut.
4. Membuat list kosong yang akan berisi simpul yang telah dikunjungi.
5. Membuat list kosong yang akan diisi dengan rute pencarian dan rute solusi.
6. Push simpul awal pada stack yang telah dibuat
7. Selama stack tidak kosong dan treasure yang telah ditemui tidak sebanyak jumlah treasure pada map, maka pop stack dan simpan hasil pop sebagai simpul saat ini. Kemudian dari simpul saat ini, cari simpul sebelahnya yang dapat dilalui dan push pada stack.

8. Setelah semua treasure didapat, maka akan diperoleh rute pencarian dan rute solusinya

2. Mapping

Mapping persoalan pada BFS:

1. Simpul yang boleh dikunjungi adalah simpul dengan elemen "K", "R", dan "T".
2. Antrean nodeQueue untuk menyimpan simpul yang belum dan akan dikunjungi.
3. Prioritas arah simpul yang dibangkitkan adalah L R U D
4. Daftar visited untuk menyimpan simpul yang telah dikunjungi.
5. Terdapat list rute solusi dan rute pencarian untuk menyimpan sekaligus sebagai hasil return algoritma

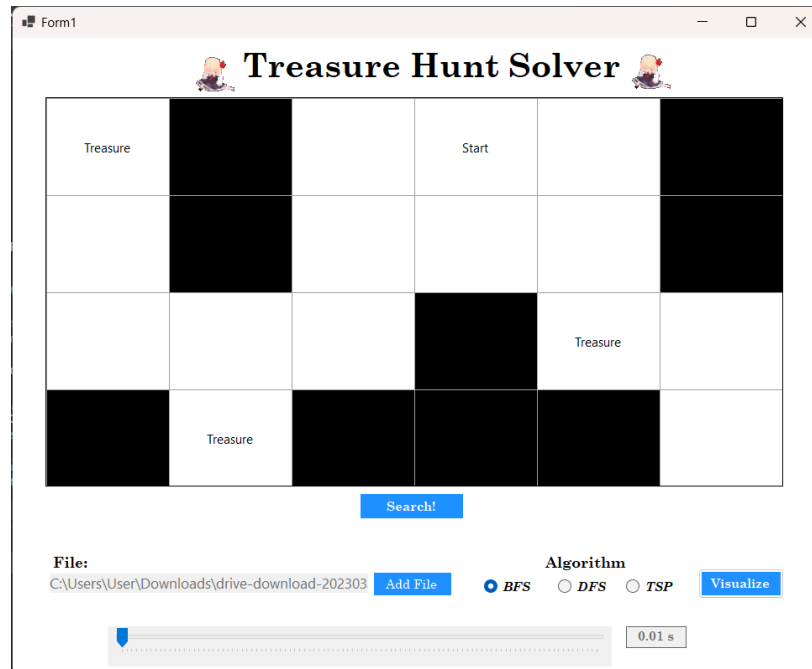
Mapping persoalan pada DFS:

1. Simpul yang boleh dikunjungi adalah simpul dengan elemen "K", "R", dan "T".
2. Stack stackNode untuk menyimpan simpul yang belum dan akan dikunjungi.
3. Prioritas arah simpul yang dibangkitkan berkebalikan dengan prioritas arah simpul BFS, yaitu D U R L.
4. Daftar visited untuk menyimpan simpul yang telah dikunjungi
5. Terdapat list rute solusi dan rute pencarian untuk menyimpan sekaligus sebagai hasil return algoritma

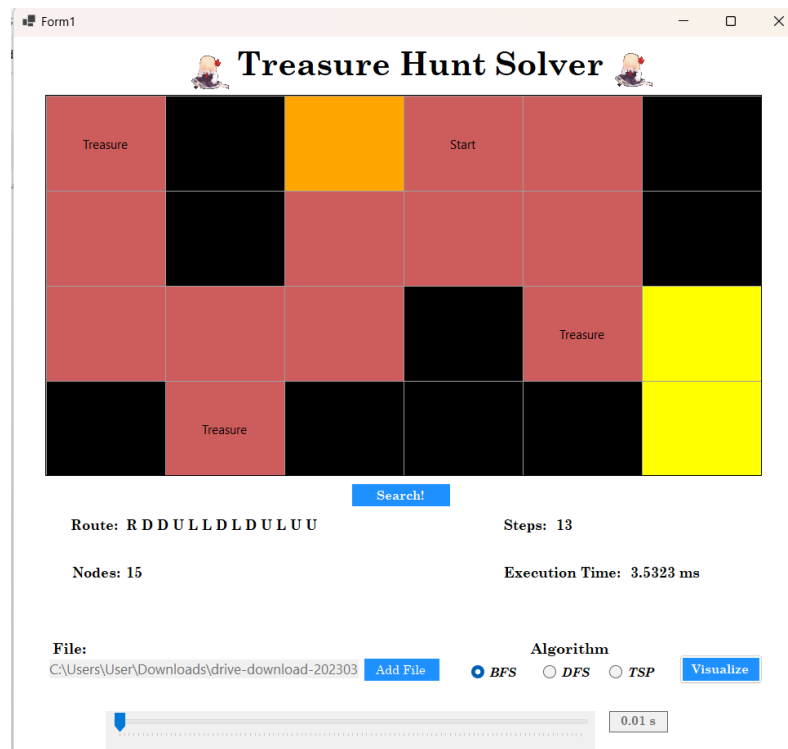
3. Ilustrasi Kasus

Contoh kasus 1:

- a) Contoh masukan:



b) Contoh Keluaran:



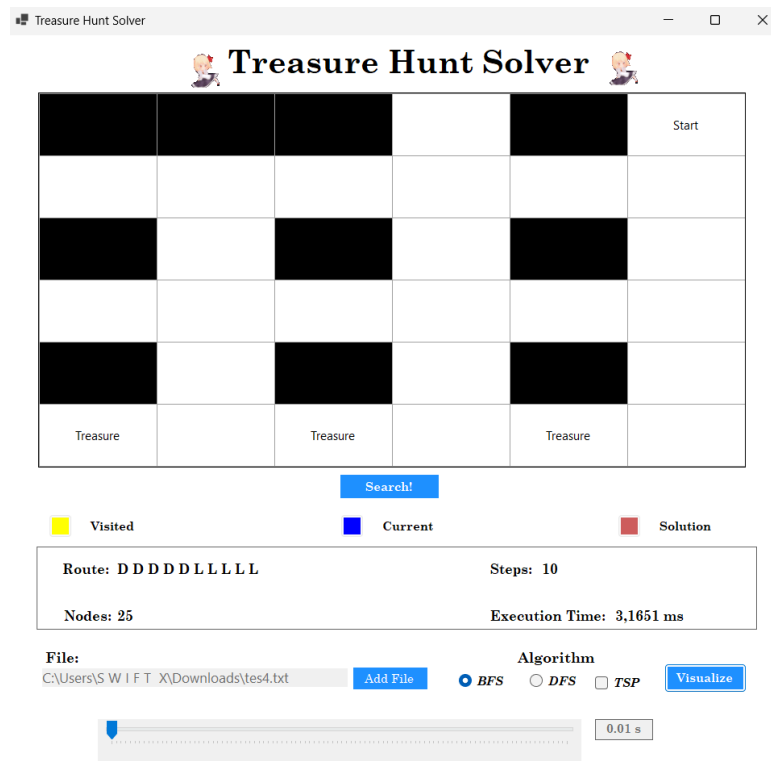
Contoh kasus 2:

a) Contoh masukan

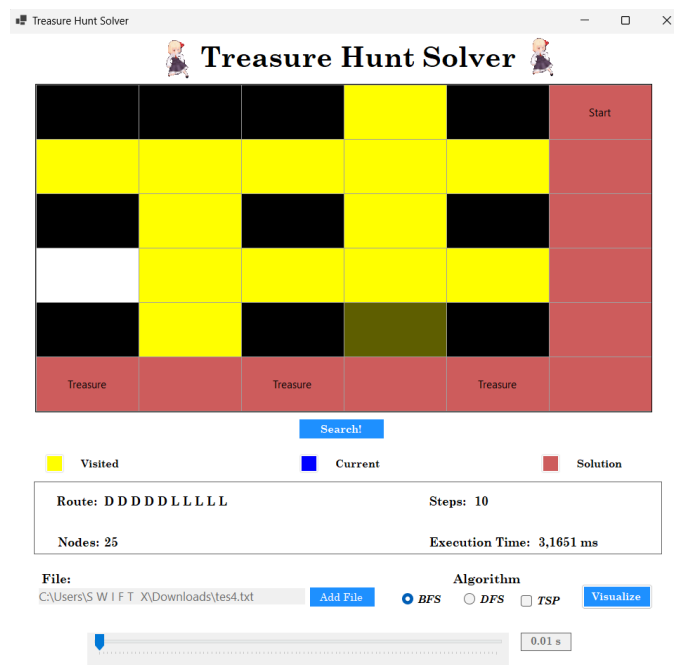
b) Contoh keluaran

Contoh kasus 3:

a) Contoh masukan



b) Contoh keluaran



BAB IV

ANALISIS PEMECAHAN MASALAH

1. Implementasi Program

1.1. Bfs.cs

```

startingPoint <- getStartingPoint();
nodeQueue<-{};
totalTreasure <- getTreasureAmount();
visited <- {startingPoint};
currentTreasure <- 0;
while (currentTreasure<totalTreasure):
    while (nodeQueue tidak kosong):
        temp <- dequeue(nodeQueue);
        currentNode <- temp[0];
        path <- temp[1];
        if (map[currentNode[0],currentNode[1]]=="T")
            currentTreasure <- currentTreasure+1;
            startingPoint <- currentNode;
            break;
        else
            i traversal 0..4:
                nextNode <- getNextNode(currentNode,i);
                if (nextNode belum dikunjungi dan
nextNode != "X")
                    enqueue(nodeQueue,
<nextNode,path+currentNode>);
                    add(visited, nextNode);
            clear(nodeQueue);
            enqueue(nodeQueue,<startingPoint,path>)
-> <path,visited>

```

1.2. Dfs.cs

```

maze <- this.map.getMap();

```

```

startingPoint <- this.map.getStartingPoint();
nTreasure <- this.map.getTreasureAmount();
nodeStack <- {};
solution <- {};
searchingRoute <- {};
visited = {};

treasureGet <- 0;

nodeStack.Push(new List<int[]> { startingPoint });

while (nodeStack.Count > 0 && treasureGet !=
nTreasure)
{
    currentTop <- nodeStack.Pop();
    currentNode <- currentTop[currentTop.Count - 1];
    temp <- currentTop.ToList();

    visited.Add(currentNode);

    if (!isBranched(maze, visited, currentNode))
    {
        searchingRoute.Add(currentTop);
    }

    if (maze[currentNode[0], currentNode[1]] == "T")
    {
        solution.Add(currentTop);
        treasureGet++;
    }

    i traversal 0..4
    {
        nextNode <- getNextNode(currentNode, i);
        if (maze[nextNode[0], nextNode[1]] != "X" &&
!isVisited(visited, nextNode))
        {
            temp.Add(nextNode);
            nodeStack.Push(new List<int[]>(temp));
            temp.RemoveAt(temp.Count - 1);

```

```

        }
    }
    temp.Clear();
}

if (!searchingRoute.Contains(currentTop))
{
    searchingRoute.Add(currentTop);
}

if (treasureGet != nTreasure)
{
    solved = false;
}
else solved = true;

-> createRoute(solution),
createRoute(searchingRoute), solved;

```

2. Struktur Data

1. Kelas Solver

Atribut kelas solver terdiri atas map. atribut map merupakan objek yang menyimpan matriks dari map yang dibaca dari file input. Method yang ada pada kelas Solver adalah getNextNode yang mengembalikan node selanjutnya, siVisited yang mengembalikan apakah node B bersebelahan dengan node A, isBranched yang mengembalikan apakah suatu simpul mempunyai cabang, createRoute yang mengembalikan rute dari titik awal ke treasure terakhir, heuristicDirection yang mengembalikan urutan arah dan optimizedSolution yang mengembalikan hasil mengoptimasi solusi yang didapat.

2. Kelas Bfs

Kelas Bfs berisi method bfsearch yang melakukan pencarian “Treasure” sesuai dengan kaidah BFS dan mengembalikan daftar simpul solusi, daftar simpul yang dikunjungi.

3. Kelas Dfs

Kelas Dfs berisi method dfsearch yang melakukan pencarian “Treasure” sesuai dengan kaidah DFS dan mengembalikan daftar simpul solusi, daftar simpul yang dikunjungi.

4. Kelas Tsp

Kelas Tsp berisi method tspproblem yang menyelesaikan permasalahan tsp dan mengembalikan daftar simpul solusi dan daftar simpul yang dikunjungi, getRouteBack yang mengembalikan rute ke titik start dari treasure terakhir, tspwithbfs yang mengembalikan solusi permasalahan tsp dengan algoritma BFS, dan tspwithdfs yang mengembalikan solusi permasalahan tsp dengan algoritma DFS.

5. Kelas Map

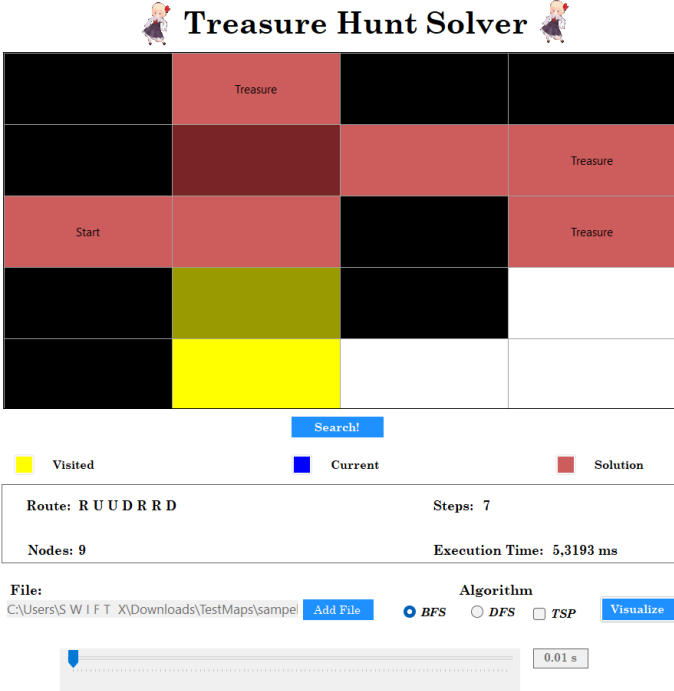
Atribut kelas Map terdiri atas map, startingPoint, dan treasureCount. atribut map merupakan array of string yang menyimpan peta yang dibaca dari file, atribut startingPoint menyimpan titik awal, dan treasureCount menyimpan banyaknya treasure yang ada. Kelas Map berisi method setMap yang merubah element map pada suatu index, getMap yang mengembalikan map, showMap yang menampilkan map di console(untuk debug), getStartingPoint yang mengembalikan startingPoint, dan getTreasureAmount yang mengembalikan treasureCount.

3. Tata Cara Penggunaan Program

1. Jalankan .exe yang ada di dalam folder bin
2. Pilih file peta yang ingin dilakukan pencarian
3. Klik tombol visualize
4. Pilih algoritma yang ingin dijalankan
5. Klik tombol tsp untuk menampilkan solusi tsp juga (opsional)
6. Klik tombol search untuk menampilkan solusi

4. Hasil Pengujian

1. TestMap Sampel 1



Treasure Hunt Solver

	Treasure		
			Treasure
Start			Treasure

Search!

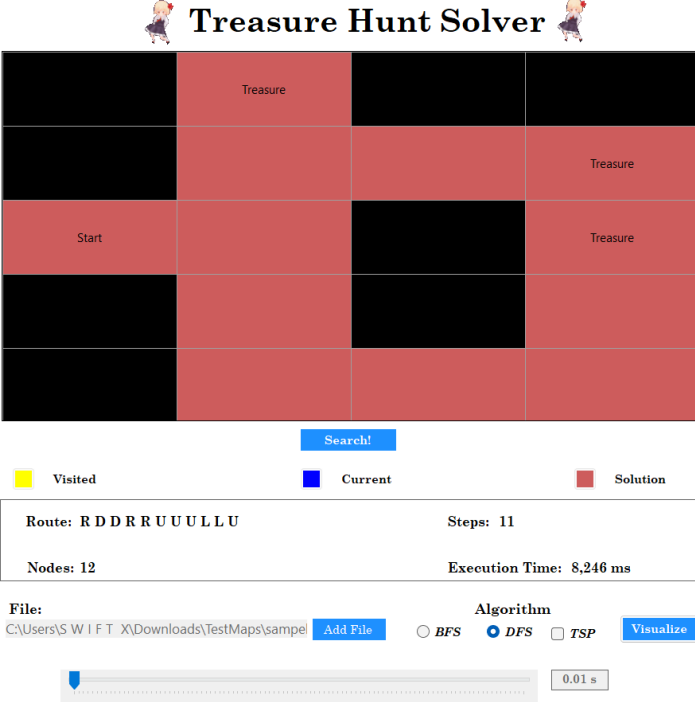
■ Visited
 ■ Current
 ■ Solution

Route: R U D R R D Steps: 7
 Nodes: 9 Execution Time: 5,3193 ms

File: C:\Users\S W I F T \X\Downloads\TestMaps\sampel Add File
 Algorithm: ☒ BFS ☐ DFS ☐ TSP Visualize

0.01 s

Sampel 1 BFS



Treasure Hunt Solver

	Treasure		
			Treasure
Start			Treasure

Search!

■ Visited
 ■ Current
 ■ Solution

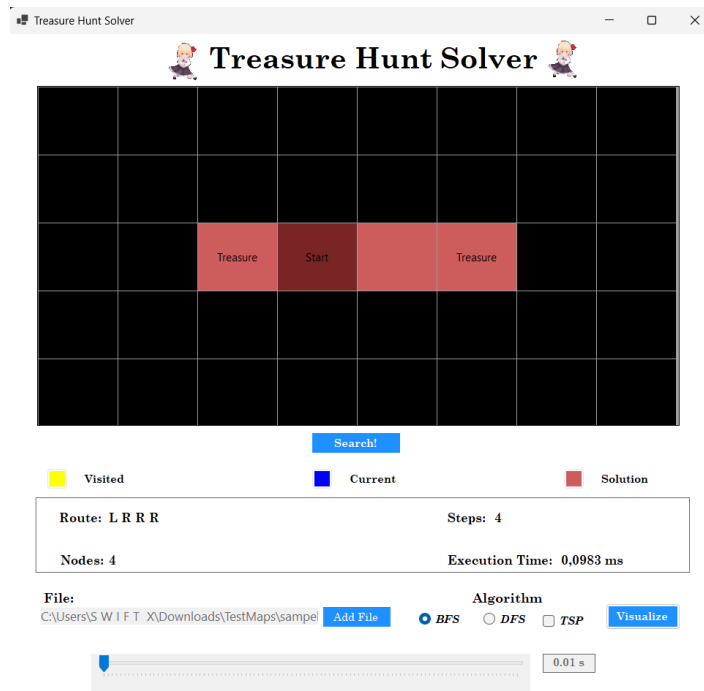
Route: R D D R R U U U L L U Steps: 11
 Nodes: 12 Execution Time: 8,246 ms

File: C:\Users\S W I F T \X\Downloads\TestMaps\sampe Add File
 Algorithm: ☐ BFS ☒ DFS ☐ TSP Visualize

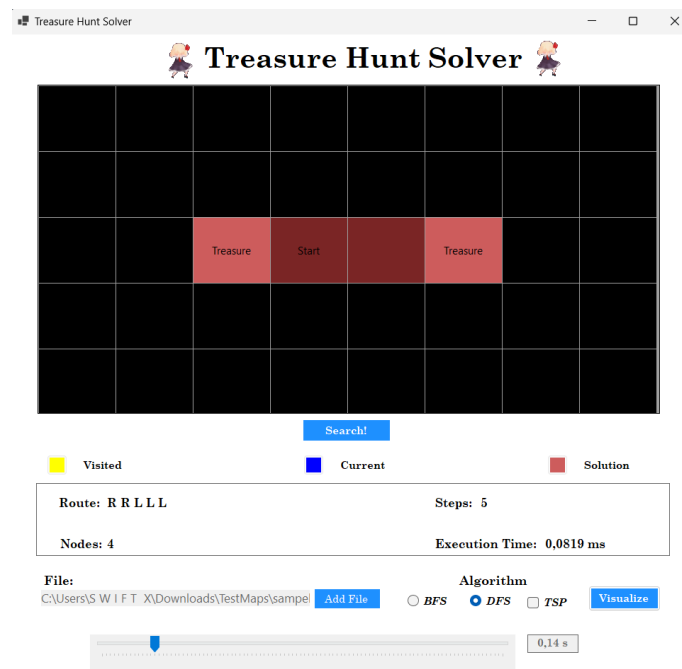
0.01 s

Sampel 1 DFS

2. TestMap Sampel 2

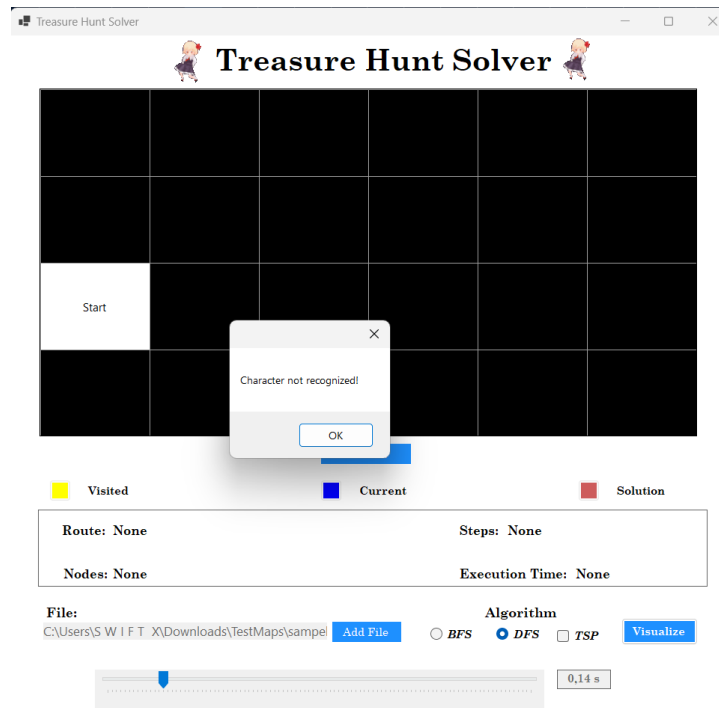


Sampel 2 BFS



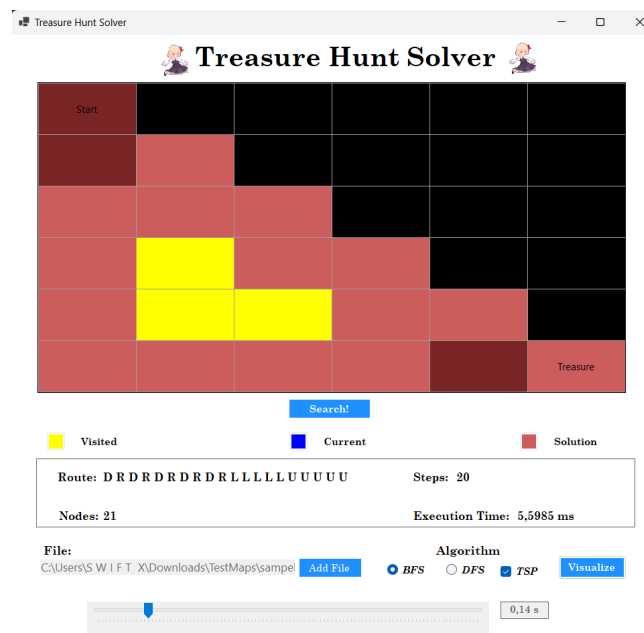
Sampel 2 DFS

3. TestMap Sampel 3

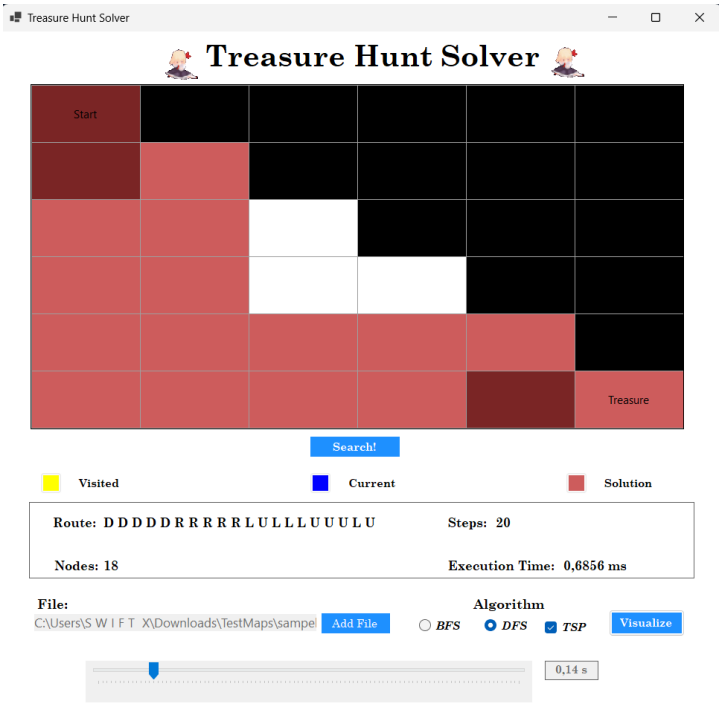


Sampel 3 Map Tidak Valid

4. TestMap Sampel 4

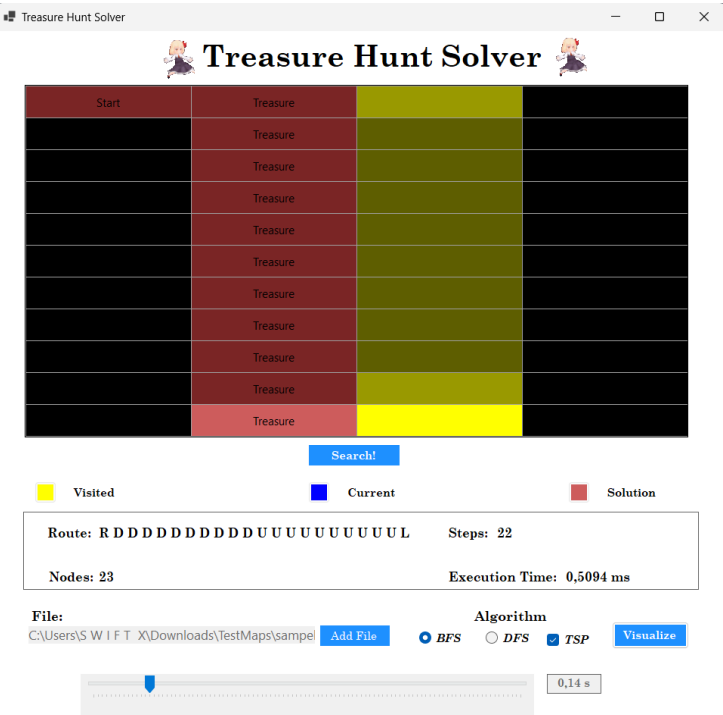


Sampel 4 TSP BFS

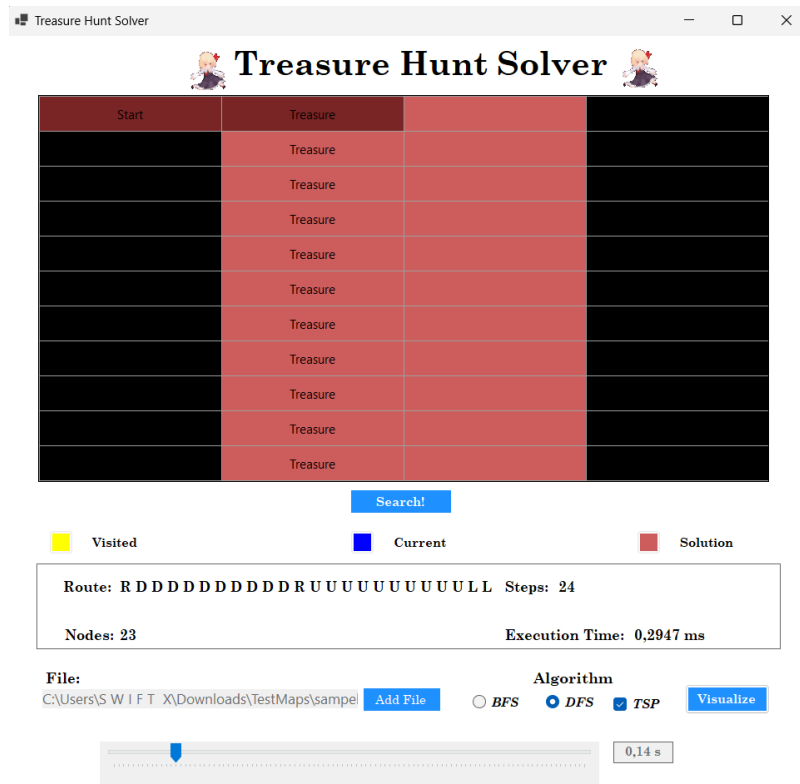


Sampel 4 TSP DFS

5. TestMap Sampel 5



Sampel 5 TSP BFS



Sampel 5 TSP DFS

5. Analisis Solusi

Perbedaan terbesar di antara BFS dan DFS adalah bagaimana algoritma menelusuri graf. BFS mengiterasi semua simpul yang bertetangga dengan simpul yang sedang dikunjungi, sementara itu DFS menelusuri semua simpul yang terhubung ke simpul awal sampai tidak ada lagi simpul yang bisa dikunjungi.

BFS lebih menguntungkan dibanding DFS ketika treasure terletak secara tersebar dan arah nya saling berlawanan karena sifat BFS yang mengiterasi semua simpul yang bertetangga.

DFS lebih menguntungkan dibanding BFS ketika treasure terletak di satu arah yang sama, karena sifat DFS yang akan mendalami suatu simpul terlebih dahulu.

Dari hasil pengujian sampel 1 dan 2, terlihat bahwa BFS lebih optimal dari DFS. Hal ini dapat dilihat dari jumlah step pada solusi yang dihasilkan. Namun jika diperhatikan, letak treasure pada map berada pada arah yang sama sehingga seharusnya hal ini lebih menguntungkan DFS. Penyebab utama dari solusi DFS tidak optimal adalah dari prioritas

arah gerak yang berbeda dengan BFS. Pada program ini, BFS memiliki prioritas arah gerak L U R D, sedangkan DFS memiliki prioritas arah gerak D R U L. Akibatnya dengan sampel 1 treasure terdekatnya berada di atas dan sampel 2 yang di sebelah kiri, maka akan lebih menguntungkan bagi BFS dalam menentukan solusi.

Pada hasil pengujian sampel 3, file txt yang diinput hanya memiliki banyak kesalahan simbol. Dengan validasi input, simbol-simbol yang salah tersebut dianggap sebagai X (wall) dan tetap bisa divisualisasikan. Namun ketika pencarian solusi dilakukan, maka akan muncul pesan error pada sebuah messagebox dan pencarian tidak dapat dilakukan.

Pengujian sampel 4 dan 5 dilakukan dengan mencari solusi TSP. Untuk sampel 4 solusi dengan DFS lebih optimal, meskipun step solusi sama namun execution time lebih cepat untuk DFS. Sedangkan untuk sampel 5 solusi dengan BFS lebih optimal secara step, sedangkan DFS lebih optimal secara execution time. Perbedaan solusi ini dapat disebabkan oleh pencarian seluruh treasure dengan BFS maupun DFS dan juga banyak simpul yang telah dilewati selama pencarian. Algoritma TSP pada program ini memprioritaskan simpul yang belum pernah dikunjungi, oleh sebab itu step solusi dapat menjadi lebih panjang bila masih banyak simpul yang belum dikunjungi.

BAB V

KESIMPULAN DAN SARAN

1. Kesimpulan

Kesimpulan yang diperoleh adalah kedua metode pencarian memiliki kelebihan dan kekurangannya masing-masing bergantung pada graf yang ditelusuri. BFS mengutamakan pencarian melebar sehingga bila treasure terletak secara tersebar dan arahnya saling berlawanan, maka dengan metode BFS ini pencarian seluruh treasure akan lebih optimal. Sedangkan DFS mengutamakan pencarian mendalam sehingga pada kasus dimana treasure terletak di 1 arah yang sama, maka metode ini lebih optimal untuk menemukan semua treasure yang ada.

2. Saran

Saran untuk pengembangan lebih lanjut adalah untuk meningkatkan kreativitas dalam tampilan antarmuka aplikasi. Selain itu, rute pencarian dan rute solusi dapat lebih dioptimasi.

3. Refleksi

Melalui tugas besar ini, penulis belajar mengenai banyak hal. Dalam mengerjakan tugas besar, penulis sadar bahwa diperlukan energi dan waktu yang banyak sehingga dalam mengerjakan sebaiknya tidak banyak ditunda dan dikerjakan secepatnya. Pembagian tugas yang merata dan adil juga penting dalam kerja kelompok, karena dengan begitu, pengerjaan menjadi lebih efektif dan sinkron karena sesuai dengan kemampuan masing-masing anggota serta jumlah beban terbagi rata. Dalam mendesain aplikasi tugas besar ini penulis juga sadar bahwa penulis masih kurang pengalaman dalam mendesain GUI.

4. Tanggapan

Penulis berkesan bahwa tugas besar 2 Strategi Algoritma yang telah diselesaikan ini cukup menyenangkan dan menantang. Tugas besar 2 ini juga melatih dan membuat

penulis belajar pengalaman baru untuk dapat mendesain suatu layout aplikasi serta mempelajari lebih dalam bahasa pemrograman C#.

DAFTAR PUSTAKA

- Ginting, S.E. and Sembiring, A.S. (2019) Comparison of breadth First Search (BFS) and depth-first search (DFS) methods on file search in structure directory Windows, Login. SEAN Institute. Available at: <https://www.neliti.com/publications/326767/comparison-of-breadth-first-search-bfs-and-depth-first-search-dfs-methods-on-fil> .

LAMPIRAN

Link Github : https://github.com/haidarhamda/Tubes2_DuelisSejati

Link video Youtube: <https://youtu.be/Cv4wnb4kvRE>