

Tugas Besar 2 IF2211 Strategi Algoritma

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

Disusun oleh:

Althaaf Khasyi Atisomya 13521130

Dhanika Novlisariyanti 13521132

Brigita Tri Carolina 13521156



INSTITUT TEKNOLOGI BANDUNG

TEKNIK INFORMATIKA

2023

DAFTAR ISI

DAFTAR ISI.....	2
BAB I.....	5
1.1 Latar Belakang	5
1.2 Deskripsi Tugas.....	5
.....	8
Gambar 1 Tampilan Program Sebelum dicari Solusinya.....	8
.....	8
Gambar 2 Tampilan Program Setelah dicari Solusinya.....	8
BAB II.....	10
2.1 Graf Traversal	10
2.2 Algoritma BFS (Breadth-First-Search)	10
2.3 Algoritma DFS (Depth-First Search)	11
2.4 C# GUI Development	13
BAB III	14
3.1 Algoritma Penyelesaian Permasalahan Maze Treasure Hunt dengan Pendekatan BFS (Breadth-First-Search)	14
3.2 Algoritma Penyelesaian Permasalahan Maze Treasure Hunt dengan Pendekatan DFS (Depth-First Search).....	15
3.3 Mapping Persoalan Maze Treasure Hunt dengan Pendekatan BFS.....	15
3.4 Mapping Persoalan Maze Treasure Hunt dengan Pendekatan DFS.....	16
3.5 Contoh Ilustrasi Kasus Menggunakan Algoritma Lain(?)	16
BAB IV	18
4.1 Implementasi Program.....	18
a. bfs.cs.....	18
b. dfs.cs	20

4.2 Struktur Data dan Spesifikasi Program.....	23
4.3 Tata Cara Penggunaan Program.....	25
Gambar 4.3.1 Home Screen	25
Gambar 4.3.2 Menu Screen	26
Gambar 4.3.3 Menu Screen	26
Gambar 4.3.3 Menu Screen	27
4.4 Hasil Pengujian	27
Gambar 4.4.1.1 Sampel-1.txt	27
Gambar 4.4.1.2 Sampel-1 BFS	27
Gambar 4.4.1.3 Sampel-1 BFS with TSP	28
Gambar 4.4.1.4 Sampel-1 DFS	28
Gambar 4.4.1.5 Sampel-1 DFS with TSP	28
Gambar 4.4.2.1 Sampel-2.txt	29
Gambar 4.4.2.2 Sampel-2 BFS	29
Gambar 4.4.2.3 Sampel-2 BFS with TSP	30
Gambar 4.4.2.4 Sampel-2 DFS	30
Gambar 4.4.2.5 Sampel-2 DFS with TSP	30
Gambar 4.4.3.1 Sampel-3.txt	31
Gambar 4.4.3.2 Sampel-3	31
Gambar 4.4.4.1 Sampel-4.txt	31
Gambar 4.4.4.2 Sampel-4 BFS	32
Gambar 4.4.4.3 Sampel-4 BFS with TSP	32
Gambar 4.4.4.4 Sampel-4 DFS	33
Gambar 4.4.4.4 Sampel-4 DFS with TSP	33
Gambar 4.4.5.1 Sampel-5.txt	34
Gambar 4.4.5.1 BFS.....	34
Gambar 4.4.5.2 Sampel-5 BFS with TSP	35

Gambar 4.4.5.3 Sampel-5 DFS	35
Gambar 4.4.5.1 Sampel-5 DFS with TSP	35
4.5 Analisis Desain Solusi Algoritma BFS dan DFS.....	35
BAB V	37
5.1 Kesimpulan	37
5.2 Saran	37
5.3 Refleksi	37
5.4 Tanggapan.....	38
DAFTAR PUSTAKA	39
LINK REPOSITORY	40
LINK VIDEO.....	40

BAB I

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.

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, 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

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.

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.

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.

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. 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.

Spesifikasi Program:

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

Treasure Hunt Solver

Input

Filename

Algoritma
☒ BFS
☐ DFS

Output

Start			
			Treasure
	Treasure		

Route:
Steps:

Nodes :
Execution Time :

Gambar 1 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 Tampilan Program Setelah dicari Solusinya

Spesifikasi GUI:

- 1) Masukan program adalah file maze treasure hunt tersebut atau nama filenya.
- 2) Program dapat menampilkan visualisasi dari input file maze dalam bentuk grid dan pewarnaan sesuai deskripsi tugas.
- 3) Program memiliki toggle untuk menggunakan alternatif algoritma BFS ataupun DFS.

- 4) Program memiliki tombol search yang dapat mengeksekusi pencarian rute dengan algoritma yang bersesuaian, kemudian memberikan warna kepada rute solusi output.
- 5) Luaran program adalah banyaknya node (grid) yang diperiksa, banyaknya langkah, rute solusinya, dan waktu eksekusi algoritma.
- 6) (Bonus) Program dapat menampilkan progress pencarian grid dengan algoritma yang bersesuaian. Hal tersebut dilakukan dengan memberikan slider / input box untuk menerima durasi jeda tiap step, kemudian memberikan warna kuning untuk tiap grid yang sudah diperiksa dan biru untuk grid yang sedang diperiksa.
- 7) (Bonus) Program membuat toggle tambahan untuk persoalan TSP. Jadi apabila toggle dinyalakan, rute solusi yang diperoleh juga harus kembali ke titik awal setelah menemukan segala harta karunnya (Tetap dengan algoritma BFS atau DFS).
- 8) GUI dapat dibuat sekreatif mungkin asalkan memuat 5 (7 jika mengerjakan bonus) spesifikasi di atas.

Program yang dibuat harus memenuhi spesifikasi wajib sebagai berikut:

- 1) Buatlah program dalam bahasa C# untuk mengimplementasi Treasure HuntSolver sehingga diperoleh output yang diinginkan. Penelusuran harus memanfaatkan algoritma BFS dan DFS.
- 2) Awalnya program menerima file atau nama file maze treasure hunt.
- 3) Apabila filename tersebut ada, Program akan melakukan validasi dari file input tersebut. Validasi dilakukan dengan memeriksa apakah tiap komponen input hanya berupa K, T, R, X. Apabila validasi gagal, program akan memunculkan pesan bahwa file tidak valid. Apabila validasi berhasil, program akan menampilkan visualisasi awal dari maze treasure hunt.
- 4) Pengguna memilih algoritma yang digunakan menggunakan toggle yang tersedia.
- 5) Program kemudian dapat menampilkan visualisasi akhir dari maze (dengan pewarnaan rute solusi).
- 6) Program menampilkan luaran berupa durasi eksekusi, rute solusi, banyaknya langkah, serta banyaknya node yang diperiksa.

BAB II

Landasan Teori

2.1 Graf Traversal

Algoritma traversal graf adalah mengunjungi simpul dengan cara yang sistematis. Di antaranya dengan:

1. Pencarian melebar (*breadth first search*)
2. Pencarian mendalam (*depth-first-search*)
3. Asumsi: graf terhubung

Pada kasus ini, graf adalah representasi dari sebuah persoalan. Dilakukan traversal graf untuk mendapatkan sebuah solusi. Dalam proses pencarian solusi, terdapat dua pendekatan:

1. Graf statis: graf yang sudah terbentuk sebelum proses pencarian dilakukan
2. Graf dinamis: graf yang terbentuk saat proses pencarian

Pada *Maze Treasure Hunt*, graf yang digunakan adalah graf dinamis karena graf terbentuk pada saat proses pencarian solusi.

2.2 Algoritma BFS (Breadth-First-Search)

Algoritma BFS (Breadth-First-Search) adalah algoritma pencarian solusi dengan sebuah graf atau pohon yang digunakan untuk mencari semua simpul yang dapat dibangkitkan dengan cara menjelajahi graf secara melebar atau secara horizontal terlebih dahulu sebelum melakukan pencarian secara mendalam atau vertikal.

Algoritma BFS bekerja dengan cara mengeksplorasi semua simpul yang bertetangga dengan simpul awal, dilanjutkan dengan mengeksplorasi simpul yang bertetangga dengan simpul tersebut, dan seterusnya hingga semua simpul dapat dicapai atau dalam hal pencarian solusi, sudah dicapai sebuah daun simpul yang merupakan solusi dari permasalahan.

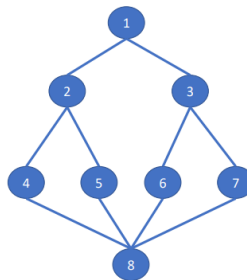
Algoritma BFS biasa digunakan untuk memecahkan masalah seperti mencari jalur untuk mencapai suatu tempat, mencari jalur terpendek dari suatu simpul ke simpul lain, dan permasalahan-permasalahan yang menyangkut penjelajahan menggunakan graf secara sistematis dan terstruktur. Berikut adalah *pseudocode* dari prosedur breadth-first-search secara umum:

```

procedure BFS (input v: integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian BFS
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi ditulis ke layer}
Deklarasi
    w: integer
Algoritma:
    for w=1 to n do
        visited[v] = false
    visited[s] = true
    queue.add(s)
    Write(s)
    while queue is not empty
        dequeue(u) from queue
        Write(u)
        for each neighbor v of u
            if visited[v] is false
                visited[v] = true
                queue.add(v)

```

Ilustrasi pencarian secara breadth-first search pada graf traversal:



Hasil pencarian dari simpul 1:

1, 2, 3, 4, 5, 6, 7, 8

2.3 Algoritma DFS (Depth-First Search)

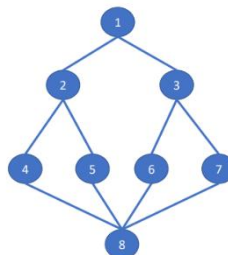
Algoritma DFS (Depth-First Search) adalah algoritma pencarian solusi dengan sebuah graf atau pohon yang digunakan untuk mencari semua simpul yang dapat dibangkitkan dengan cara menelusuri suatu jalur secara vertikal ke bawah terlebih dahulu, kemudian dilanjutkan ke jalur yang belum dikunjungi secara horizontal. Algoritma DFS mengunjungi setiap simpul secara mendalam sebelum berpindah ke simpul berikutnya.

Secara sederhana, algoritma pencarian Depth-First Search akan memulai penelusuran dari simpul v dilanjutkan dengan penelusuran pada simpul w yang merupakan tetangga dari simpul v. Ulangi algoritma DFS pada simpul w. Ketika mencapai simpul yang semua

tetangganya telah dikunjungi pencarian akan *backtrack* ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul tetangga yang belum dikunjungi. Pencarian DFS akan berakhir bila tidak ada lagi simpul yang telah dikunjungi. Berikut adalah pseudocode dari prosedur depth-first search secara umum.

```
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 layer}
Deklarasi
    w: integer
Algoritma:
    write(v)
    visited[v] = true
    for w=1 to n do
        if A[v,w] = 1 then {v bertentagga dengan w}
            if not visited[w] then
                DFS(w)
```

Ilustrasi pencarian secara depth-first dearch (DFS) pada graf traversal:



Hasil pencarian dari simpul 1:

1, 2, 4, 8, 5, 6, 3, 7

Salah satu penerapan algoritma DFS dapat dilakukan dengan memanfaatkan stack, stack akan digunakan untuk menyimpan simpul-simpul yang akan dikunjungi sesuai urutan stack (masuk terakhir-keluar pertama). Berikut merupakan Langkah Langkah algoritma DFS dengan memanfaatkan stack:

1. Masukkan simpul awal v pada stack
2. Ambil simpul dari tumpukan stack teratas periksa apakah simpul merupakan solusi.
3. Jika simpul merupaka solusi, pencarian selesai.

4. Jika simpul bukan merupakan solusi, masukkan seluruh simpul yang bertetangga dengan simpul v ke dalam stack.
5. Ulangi pencarian dari Langkah

2.4 C# GUI Development

Pada C# dengan menggunakan bantuan Visual Studio, developer dapat membuat aplikasi dengan *Graphical User Interface(GUI)* dengan menggunakan .NET framework. Visual Studio menyediakan berbagai banyak framework mulai dari WinForm App, Windows Presentation Foundation, dan lain-lain. Penggunaan framework ini bergantung pada kebutuhan masing-masing program. Pada program ini, penulis menggunakan WinForm App.

BAB III

Penjelasan Algoritma

3.1 Algoritma Penyelesaian Permasalahan Maze Treasure Hunt dengan Pendekatan BFS (Breadth-First-Search)

Pada penyelesaian permainan Maze Treasure Hunt ini digunakan salah satunya algoritma dengan pendekatan *Breadth-First-Search* atau BFS. Goal pada Maze Treasure Hunt ini adalah untuk mencari jalur dari *starting point* menuju ke *treasure-treasure* yang ada pada *maze*. Langkah-langkah penyelesaian menggunakan metode BFS adalah sebagai berikut.

1. Inisialisasi *queue* dan tambahkan *starting point* ke *queue*.
2. Inisialisasi sebuah *hash map* untuk menyimpan *parent* dari cabang yang dibangkitkan.
3. Set *parent* dari *starting point* sebagai null.
4. Inisialisasi sebuah *list of path* untuk menyimpan index jalur menuju *treasure*.
5. Selama *queue* belum kosong maka proses sebagai berikut akan terus diulang hingga mencapai *return path*:
 - a. *Dequeue queue* yang menyimpan point atau index dari matrix tadi.
 - b. Ketika *current point* adalah *treasure* dan *treasure* yang terdapat pada maze lebih dari satu, maka tambahkan path *treasure* dari *starting point* menuju *treasure* ke sebuah *list of index*. Reset *parent* dan *queue* yang menyimpan *neighbor*, kemudian ubah *current point* menjadi *route* biasa.
 - c. Ketika *current point* adalah *treasure* dan *treasure* yang terdapat pada maze hanya 1, maka *return path* dari *starting point* menuju ke *treasure* dengan memanfaatkan *hash map* yang telah terbentuk.
 - d. Ketika *current point* bukan *treasure*, maka tambahkan *neighbor* dari *current point* pada *queue* (jika belum ada) dengan *priority right-down-left-up*. Dan set *parent* dari *neighbor* menjadi *current point*.
 - e. Kemudian proses *dequeue queue neighbor* akan dilanjutkan hingga tidak ada lagi *neighbor* yang bisa ditambahkan (*queue neighbor* 0) atau hingga semua *treasure* telah ditemukan.

3.2 Algoritma Penyelesaian Permasalahan Maze Treasure Hunt dengan Pendekatan

DFS (Depth-First Search)

Penyelesaian permasalahan *Maze Treasure Hunt* dapat diselesaikan dengan pendekatan algoritma *Depth-First Search*. Goal pada permasalahan ini adalah mencari jalur dari *starting point* menuju semua treasure yang ada. Langkah-langkah menyelesaikan permasalahan *Maze Treasure Hunt* dengan pendekatan DFS (*Depth-First Search*) adalah sebagai berikut.

1. Inisiasi stack yang menyimpan *point* (berupa indeks pada *matrix*) beserta *parent*-nya (*point*, *point parent*). Stack ini akan digunakan untuk menyimpan urutan *point* yang akan dikunjungi.
2. Inisiasi sebuah list 2D dengan nilai *false* bernama *visited* yang akan bernilai *true* apabila *point* dengan index yang sama telah dikunjungi.
3. Inisialisasi sebuah *list of path* untuk menyimpan index jalur menuju *treasure*.
4. Push *starting point* pada *stack* dengan *parent null*. Pop *stack* dan assign ke *current point*.
5. Selama *stack* belum kosong maka proses sebagai berikut akan terus diulang hingga mencapai *return path*:
 - a. Tambah *current point* pada *list of path*.
 - b. Cari semua tetangga dari *current point* yang belum dikunjungi. Push semua tetangga yang valid pada *stack* dengan *current point* sebagai *parentnya* dengan *priority right-down-left-up*.
 - c. Set *visited* dari *current point* menjadi *true*.
 - d. Apabila *current point* tidak memiliki tetangga yang valid maka akan dilakukan *backtrack*.
 - e. Pop *stack* dan assign ke *current point*.

3.3 Mapping Persoalan Maze Treasure Hunt dengan Pendekatan BFS

Mapping persoalan Maze Treasure Hunt dengan pendekatan BFS (*Breadth-First-Search*) memiliki ilustrasi sebagai berikut:

1. Pertama-tama adalah penulis menentukan simpul atau start point dari map.
2. Kemudian inisialisasi sebuah queue untuk menyimpan start point tersebut atau bisa disebut juga simpul pada graf.
3. Setelah itu, telusuri simpul lain yang merupakan tetangga dari start point, masukkan pada queue dengan urutan *right-down-left-up*.

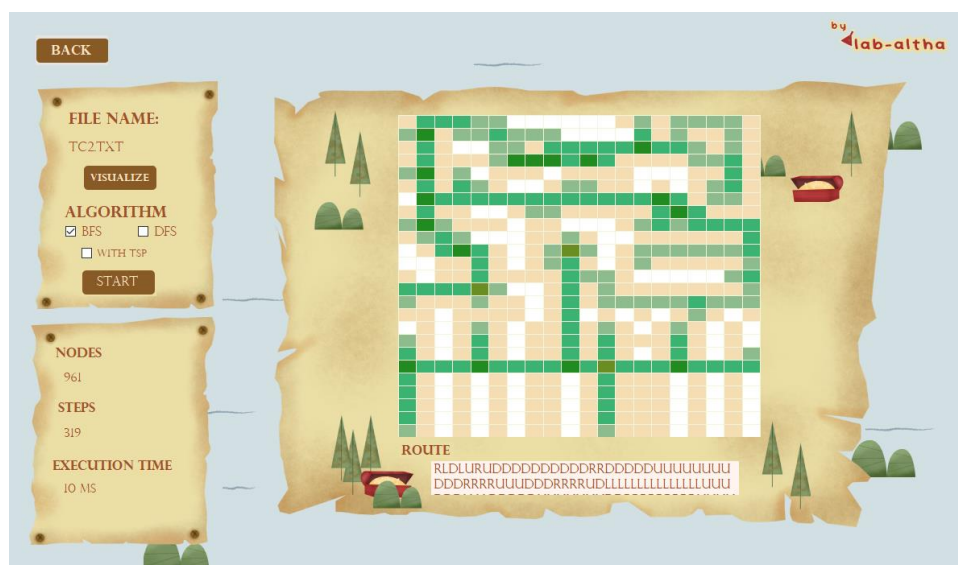
4. Maka, simpul yang ditelusuri secara melebar direpresentasikan oleh point arah-arah yang telah dimasukkan tadi ke dalam queue.
5. Pencarian dilakukan hingga sudah tidak ada lagi tetangga yang bisa ditelusuri (queue kosong) atau sudah ditemukan sebuah goal node yaitu tempat *treasure* berada.
6. Jika goal node ditemukan maka akan dikembalikan jalur dari *treasure* tersebut yaitu mulai dari simpul awal (*starting point*) hingga daun solusi.
7. Jika goal node tidak ditemukan maka dikembalikan jalur kosong atau simpul kosong.

3.4 Mapping Persoalan Maze Treasure Hunt dengan Pendekatan DFS

Mapping persoalan Maze Treasure Hunt dengan pendekatan DFS (Depth-First Search) memiliki ilustrasi sebagai berikut:

1. Tentukan simpul awal pada maze, pada persoalan Maze Treasure Hunt simpul awal ditandai dengan huruf 'K'. Push simpul awal pada stack.
2. Simpul yang terakhir masuk ke dalam stack adalah simpul yang akan dikunjungi. Pop stack sebagai simpul yang akan dikunjungi.
3. Lakukan pencarian jalur pada maze dengan menelesuri seluruh simpul tetangga yang belum pernah dikunjungi sebelumnya dari simpul yang sedang dikunjungi dengan *priority right-down-left-up*, simpan informasi tersebut pada stack.
4. Jika treasure belum ditemukan ulangi langkah 2.
5. Jika semua treasure telah ditemukan, hentikan pencarian dan jalur menuju semua treasure telah ditemukan.

3.5 Contoh Ilustrasi Kasus Menggunakan Test Case Lain



by lab-alpha

BACK

FILE NAME:
TC2.TXT
VISUALI

ALGORITHM
☐ BFS ☒ DFS
☐ WITH TSP
 START

NODES
480

STEPS
479

EXECUTION TIME
1 MS

ROUTE
 RRDDDDRRRRRRRRUULLLLLLUURRRR
 RRUUUUULLLLLRURRLUULLLLLDLULL

dasarkan hasil solusi penyelesaian *Maze Treasure Hunt* pada kasus c

Berdasarkan hasil solusi penyelesaian *Maze Treasure Hunt* pada kasus diatas, desain solusi dengan pendekatan algoritma *Breadth-First Search* (BFS) lebih efektif dibanding solusi dengan pendekatan *Depth-First Search* (DFS). Hal ini dibuktikan dengan jumlah langkah pendekatan BFS yang lebih sedikit dari penyelesaian dengan pendekatan DFS, yaitu 319 langkah dengan BFS dan 479 langkah dengan DFS. Penyelesaian permasalahan *Maze treasure Hunt* dengan pendekatan BFS membutuhkan waktu yang lebih lama dari solusi pendekatan DFS. Hal ini terjadi karena BFS akan melakukan pencarian secara meluas pada setiap level labirin dapat dilihat dari jumlah nodes penyelesaian BFS yang sangat banyak mencapai 961 *nodes*. Akan tetapi, solusi penyelesaian dengan pendekatan BFS akan menghasilkan langkah penyelesaian yang lebih pendek dari penyelesaian dengan pendekatan DFS.

BAB IV

Analisis Pemecahan Masalah

4.1 Implementasi Program

a. bfs.cs

```
bfs()
    bfsPath = List of int tuple
    bfsDirection = List of character
    bfsSteps = 0
    bfsNodes = 0
    bfsSeconds = 0

bfs(path, direction, steps, nodes, second)
    bfsPath = path
    bfsDirection = direction
    bfsSteps = steps
    bfsNodes = nodes
    bfsSeconds = second

// BFS Algorithm
bfs BFS(map)
    mapinsinde = char[map.GetLength(0), map.GetLength(1)]
    copy(map, mapinsinde, map.Length)
    countT = 0
    start = (0,0)
    for (int i = 0; i < map.GetLength(0); i++)
        for (int j = 0; j < map.GetLength(1); j++)
            if (map[i, j] = 'T')
                countT++
            if (map[i, j] = 'K')
                start = (i, j)

    // Defining direction
    directions = ((0,1), (1,0), (0,-1),(-1,0))

    // Queue for BFS
    pathqueue = queue of int tuple
    pathqueue.Enqueue(start)

    // Dictionary to store parrent of each point
    pathParent = hash map of interger tuple
    pathParent[start] = null

    count = 0
    nodesCount = 0
    steps = 0
    allPath = list of integer tuple
    tempCurrent = (0, 0)
    tempCurrent2 = (0, 0)

    s = Stopwatch()
    s.Start()
    while (pathqueue.Count > 0)
        current = pathqueue.Dequeue()
        nodesCount++

        if (mapinsinde[current.Item1, current.Item2] = 'T' && count <
countT - 1)
```

```

count++;
mapinsinde[current.Item1, current.Item2] = 'R'
path = list of integer tuple
pathqueue.Clear()
tempCurrent = current
    if (count = 1)
        while (current != null)
            path.Add(current)
            current = pathParent[current]
    else
        while (current != tempCurrent2)
            path.Add(current)
            current = pathParent[current]
path.Reverse()
foreach (solution in path)
    allPath.Add(solution)
current = tempCurrent
tempCurrent2 = current
pathParent.Clear()
pathParent = hash map of integer tuple
else if (mapinsinde[current.Item1, current.Item2] = 'T' &&
count = countT - 1)
    path = list of integer tuple
    // Construct current path to treasure
    if (count = 0)
        while (current != null)
            path.Add(current)
            current = pathParent[current]
        path.Reverse()
    else
        while (current != tempCurrent)
            path.Add(current)
            current = pathParent[current]
        path.Reverse()

        foreach (solution in path)
            allPath.Add(solution)

s.Stop()
seconds = s.ElapsedMilliseconds
steps = allPath.Count - 1
→ bfs(allPath, IndexToChar(allPath), steps,
nodesCount, seconds)
// Explore neighbours of the current point
foreach (direction in directions)
    neighbor = (current[0] + direction[0],
current[1] + direction[1])

    if (isPointValid(map, neighbor) &&
!pathParent.ContainsKey(neighbor))
        pathqueue.Enqueue(neighbor)
        pathParent[neighbor] = current

s.Stop()
second = s.ElapsedMilliseconds
→ bfs(allPath, IndexToChar(allPath), steps, nodesCount,
second)

bfs TSPWithBFS(char[,], map, Tuple<int, int> lastTreasure)
BFSAnswer = bfs.BFS(map)
maxRow = map.GetLength(0)
maxCol = map.GetLength(1)
char[,], mapinside = char[maxRow, maxCol]
Copy(map, mapinside, map.Length)

```

```

    for (int i = 0 i < maxRow i++)
        for (int j = 0 j < maxCol j++)
            if (mapinside[i, j] = 'T')
                mapinside[i, j] = 'R'
            if (mapinside[i, j] = 'K')
                mapinside[i, j] = 'T'

    mapinside[lastTreasure.Item1, lastTreasure.Item2] = 'K'
    TSP = bfs.BFS(mapinside)

    solutions = list of integer tuple
    for (int i = 0 i < BFSAnswer.bfsPath.Count i++)
        solutions.Add(BFSAnswer.bfsPath[i])
    for (int i = 1 i < TSP.bfsPath.Count i++)
        solutions.Add(TSP.bfsPath[i])

    solutionsInChar = list of char
    for (int i = 0 i < BFSAnswer.bfsDirection.Count i++)
        solutionsInChar.Add(BFSAnswer.bfsDirection[i])
    for (int i = 0 i < TSP.bfsDirection.Count i++)
        solutionsInChar.Add(TSP.bfsDirection[i])
    steps = BFSAnswer.bfsSteps + TSP.bfsSteps
    nodes = BFSAnswer.bfsNodes + TSP.bfsNodes
    → bfs(solutions, solutionsInChar, steps, nodes,
TSP.bfsSeconds)

    bool isPointValid(char[,] map, Tuple<int, int> point)
        row = point.Item1
        col = point.Item2
        if (row < 0 || row >= map.GetLength(0))
            return false
        if (col < 0 || col >= map.GetLength(1))
            return false
        if (map[row, col] = 'X')
            return false
        → true

    List of char IndexToChar(solution)
        solutionInChar = List of char
        for (int i = 0 i < Solution.Count - 1 i++)
            if (Solution[i + 1][0] - Solution[i][0] = 0 && Solution[i
+ 1][1] - Solution[i][1] = -1)
                solutionInChar.Add('L')
            else if (Solution[i + 1][0] - Solution[i][0] = 0 &&
Solution[i + 1][1] - Solution[i][1] = 1)
                solutionInChar.Add('R')
            else if (Solution[i + 1][0] - Solution[i][0] = -1 &&
Solution[i + 1][1] - Solution[i][1] = 0)
                solutionInChar.Add('U')
            else if (Solution[i + 1][0] - Solution[i][0] = 1 &&
Solution[i + 1][1] - Solution[i][1] = 0)
                solutionInChar.Add('D')
            end if
        → solutionInChar

```

b. dfs.cs

```

dfs()
    dfsPath = list of int tuple
    dfsDirection = list of char
    dfsSteps = 0

```

```

    dfsNodes = 0
    dfsSeconds = 0

dfs(path, direction, steps, nodes, second)
    dfsPath = path
    dfsDirection = direction
    dfsSteps = steps
    dfsNodes = node
    dfsSeconds = second

//DFS algorithm
dfs DFS(map)
    treasureCount = getTreasureCount(map);
    time = new System.Diagnostics.Stopwatch()

    // map max row & max column
    maxRow = length(map)
    maxCol = length(map[0])

    // mark visited points, get starting point and set visited to true
    startingPoint = getStartingPoint(map);
    for(i=0; i<maxRow; i++)
        for (j=0; j<maxCol; j++)
            visited[i][j] = false
    visited[startingPoint[0], startingPoint[1]] = true

    // create stack for DFS
    s.Push(startingPoint,null)

    // create path

    // BFS movement priority = R,D,L,U
    rowMovement = [ 0, 1, 0, -1 ]
    colMovement = [ 1, 0, -1, 0 ]

    nRow = 0, nCol = 0
    isValid = false
    currPoint = s.Pop()[0]
    // loop until all treasures found
    tima.Start()
    while (treasureCount != 0)
        path.Add(currPoint)

        // iterate all neighbors of currPoint
        for (int i = 3; i >= 0; i--)
            // neighbor row & column index
            nRow = currPoint[0] + rowMovement[i]
            nCol = currPoint[1] + colMovement[i]
            isValid = nRow >= 0 and nRow < maxRow and nCol >= 0 and
nCol < maxCol and !visited[nRow, nCol]

            // if neighbor valid: add to stack
            if (isValid and IsPointValid(map[nRow, nCol]))
                s.Push((nRow, nCol), currPoint)

```

```

        visited[currPoint[0], currPoint[1]] = true
        while (visited[s.Top()[0][0], s.Top()[0][1]])
            currPoint = s.Pop()[0]

        // treasure found
        if (map[s.Top()[0][0], s.Top()[0][1]] == 'T' &&
!visited[s.Top()[0][0], s.Top()[0][1]])
            treasureCount--

        // backtrack path's handle
        if (path.Last() != s.Top()[1])
            idx = length(path) - 2
            while (path.Last() != s.Top[1])
                search = path[idx]
                idx = path.FindIndex(x => x == search)
                path.Add(path[idx])
                idx--

        currPoint = s.Pop()[0]
        while (visited[currPoint[0], currPoint[1]])
        {
            currPoint = s.Pop()[0]
        }
        path.Add(currPoint);
        time.Stop();
        pathDirection = IndexToChar(path);
        return (path, pathDirection, pathDirection.Count(), path.Count(),
time)

// DFS with TSP
dfs TSPwithDFS(map, lastTreasure)
    dfs result = dfs.DFS(map)
    maxRow = Lenght(map)
    maxCol = length(map[0])
    mapCopy = map;
    for (int i = 0; i < maxRow; i++)
        for (int j = 0; j < maxCol; j++)
            if (mapCopy[i][j] == 'T')
                mapCopy[i][j] = 'R'
            if (mapCopy[i][j] == 'K')
                mapCopy[i][j] = 'T'

    mapCopy[lastTreasure.Item1, lastTreasure.Item2] = 'K'
    dfs tsp = dfs.DFS(mapCopy)

    // merge dfs and tsp
    nodes = result.dfsNodes + tsp.dfsNodes;
    steps = result.dfsSteps + tsp.dfsSteps;
    tsp.dfsPath.RemoveAt(0);
    return ((result.dfsPath).Concat(tsp.dfsPath),
(result.dfsDirection).Concat(tsp.dfsDirection), steps, nodes-1,
second);

```

```

bool IsPointValid(point)
    switch (point)
        case 'R':
            return true
        case 'T':
            return true
        case 'K':
            return true
        default:
            return false

```

4.2 Struktur Data dan Spesifikasi Program

Penyelesaian *Maze Treasure Hunt* dengan pendekatan algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS) menggunakan beberapa struktur data untuk merepresentasikan komponen-komponen yang akan digunakan dalam implementasinya, yaitu:

1. Map

Map/maze pada *Maze Treasure Hunt* akan direpresentasikan sebagai *list* dua dimensi dengan elemen bertipe character. Pada C# map memiliki struktur data `char[,] map`. Elemen 'K' pada *map* merupakan titik awal/*starting point*. Elemen 'T' merupakan treasure yang harus dicari. Elemen 'R' merupakan grid/point yang dapat diakses sebagai sebuah lintasan. Elemen 'X' merupakan halangan yang tidak dapat diakses sebagai jalan menuju *treasure*.

2. Point

Point pada *Maze Treasure Hunt* merepresentasikan indeks *grid* pada map yang memiliki struktur data *tuple of int,int*. Pada C# point memiliki struktur data `Tuple<int,int> point`. Indeks *row* pada point dapat diakses dengan menggunakan `point.Item1` sedangkan index *column* dapat diakses dengan `point.Item2`.

3. Queue

Point pada *Maze Treasure Hunt* pada map yang memiliki struktur data *tuple of int, int* dapat disimpan pada sebuah queue of *tuple int, int*. *Point* disimpan pada sebuah *queue* untuk memudahkan proses pengeluaran point yaitu dengan menggunakan `queue.Dequeue()` untuk mendapatkan elemen pertama yang dimasukkan pada queue. Digunakan `queue.Enqueue()` pada penambahan elemen tetangga yang ingin ditelusuri dari *current point*.

4. Stack

Stack digunakan pada penyelesaian *Maze Treasure Hunt* dengan pendekatan *Depth-First Search* (DFS) sebagai urutan point yang akan dikunjungi. Stack akan menyimpan point juga parent dari point tersebut sehingga memiliki struktur data stack of Tuple point,point. Pada C# *stack* yang digunakan pada pencarian secara DFS adalah `Stack<Tuple<Tuple<int,int>,Tuple<int,int>>>` stack. Untuk memasukan suatu elemen ke dalam *stack* dapat dilakukan dengan `stack.Push(addedElement)`, sedangkan untuk mengambil elemen pada *stack* dilakukan dengan `stack.Pop()`. `Stack.Pop()` akan mengembalikan elemen *stack* yang paling terakhir di *push*. Selain *push* dan *pop* terdapat juga `stack.Peek()` yang digunakan untuk melihat elemen stack yang paling terakhir dipush tanpa membuang elemen tersebut dari *stack*.

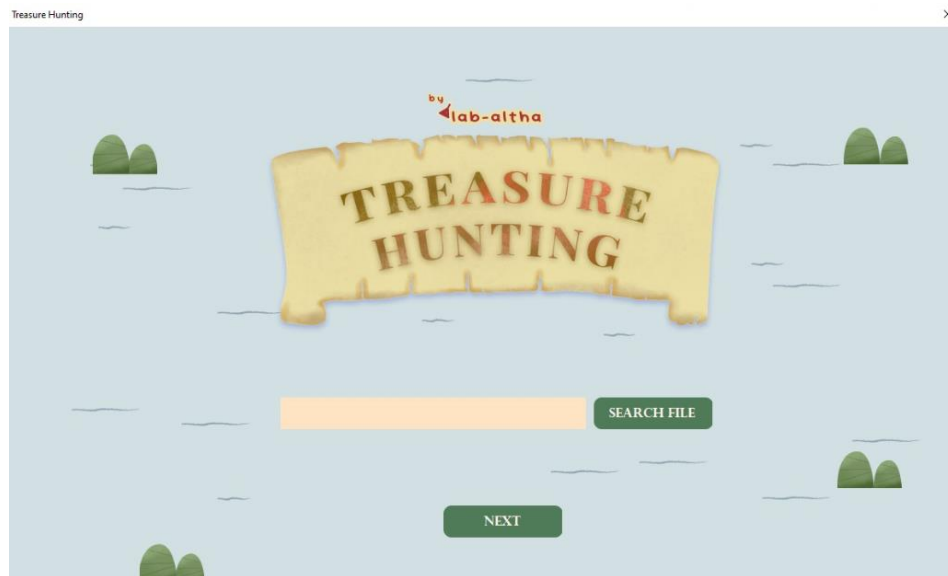
5. List of Path

Lintasan solusi permasalahan *Maze Treasure Hunt* memiliki struktur data List of point yaitu list indeks point/grid pada map yang merupakan lintasan untuk mendapatkan semua *treasure*. Pada C# list of path adalah `List<Tuple<int,int>>` path. Urutan pada list of Path merupakan solusi grid yang harus dikunjungi Mr.Krab untuk mendapatkan semua *treasure* dengan algoritma yang bersesuaian.

6. PathDirection

Urutan arah jalan pada permasalahan *Maze Treasure Hunt* direpresentasikan sebagai list of character. Pada C# Path direction memiliki struktur data `List<char>` pathDirection. Elemen character pada path direction memiliki arti arah mana Mr.Krab harus bergerak, yaitu character 'L' kearah kiri, character 'R' kearah kanan, character 'U' ke atas, dan character 'D' ke bawah.

4.3 Tata Cara Penggunaan Program



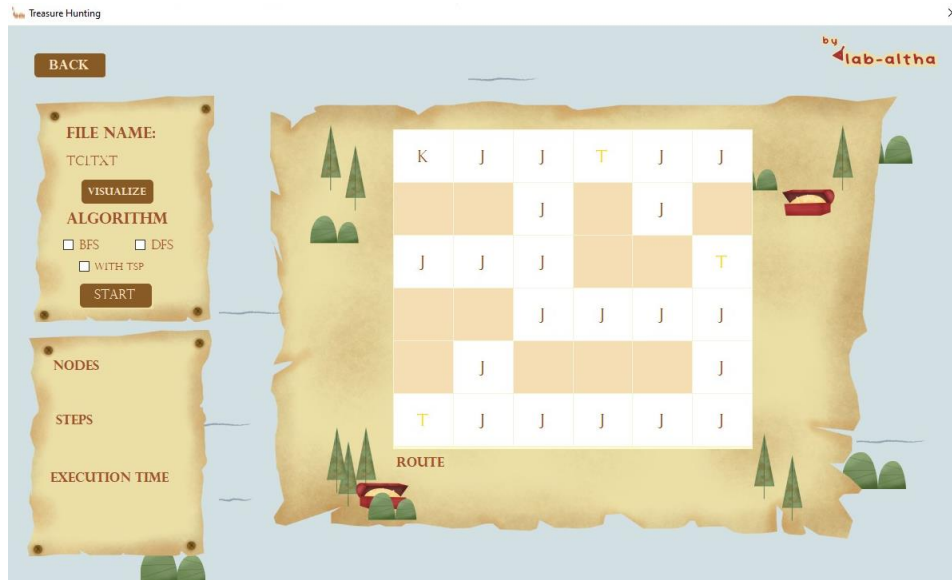
Gambar 4.3.1 Home Screen

Ketika program pertama kali dijalankan, tampilan yang paling pertama keluar adalah seperti gambar di atas. Pengguna akan diminta untuk memasukkan nama file, jika file dapat ditemukan maka antar muka akan mengeluarkan “File Found”, jika tidak ditemukan maka akan keluar “File Not Found”. Jika file sudah ditemukan, hal yang selanjutnya dilakukan adalah validasi format file. Jika format file salah maka akan keluar warning bahwa format file salah. Jika file berhasil di cek, maka pengguna dapat menekan tombol Next untuk ke menu selanjutnya.



Gambar 4.3.2 Menu Screen

Program selanjutnya akan menampilkan layar seperti gambar diatas. Pengguna diminta untuk memvisualisasikan peta terlebih dahulu sebelum melakukan pencarian treasure dengan algoritma yang disediakan. Setelah di visualisasi, tombol start akan di enabled dan dapat melakukan pencarian sesuai algoritma yang dipilih pengguna.



Gambar 4.3.3 Menu Screen

Hasil nodes, steps, execution time, dan route akan tampil jika tombol start sudah ditekan oleh pengguna dan akan ditampilkan proses rute dari peta tersebut.



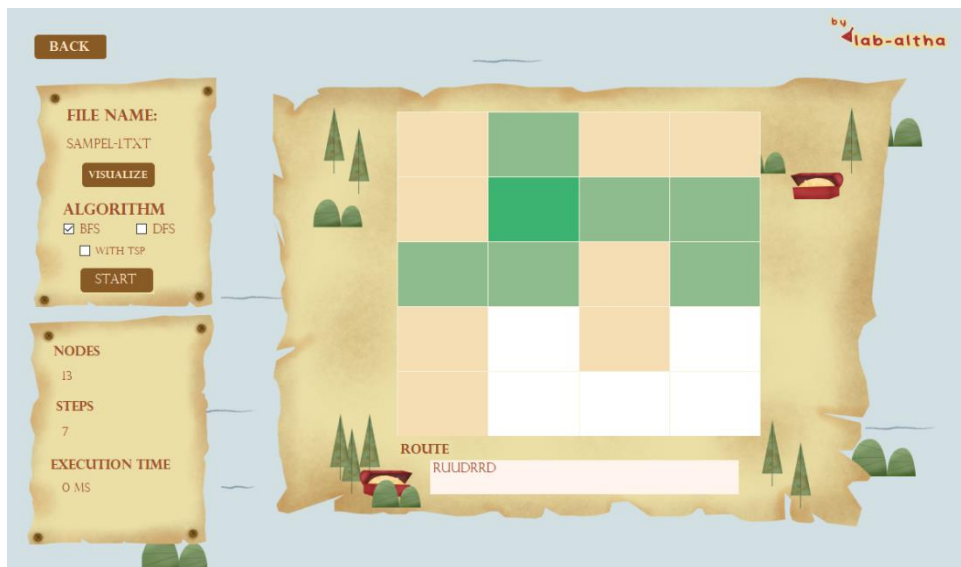
Gambar 4.3.3 Menu Screen

4.4 Hasil Pengujian

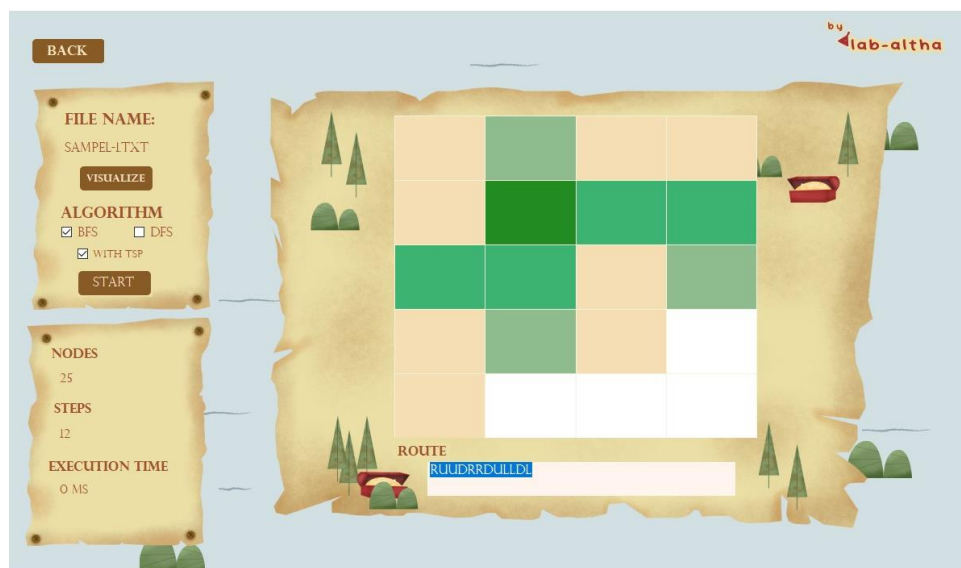
1. Sampel-1.txt

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

Gambar 4.4.1.1 Sampel-1.txt



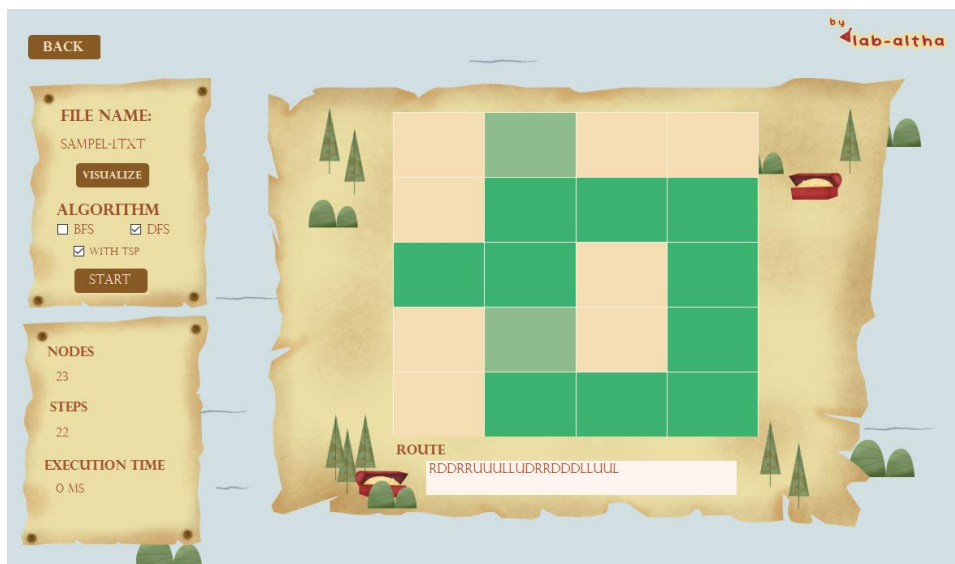
Gambar 4.4.1.2 Sampel-1 BFS



Gambar 4.4.1.3 Sampel-1 BFS with TSP



Gambar 4.4.1.4 Sampel-1 DFS



Gambar 4.4.1.5 Sampel-1 DFS with TSP

2. Sampel-2.txt

```

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

```

Gambar 4.4.2.1 Sampel-2.txt



Gambar 4.4.2.2 Sampel-2 BFS



Gambar 4.4.2.3 Sampel-2 BFS with TSP



Gambar 4.4.2.4 Sampel-2 DFS

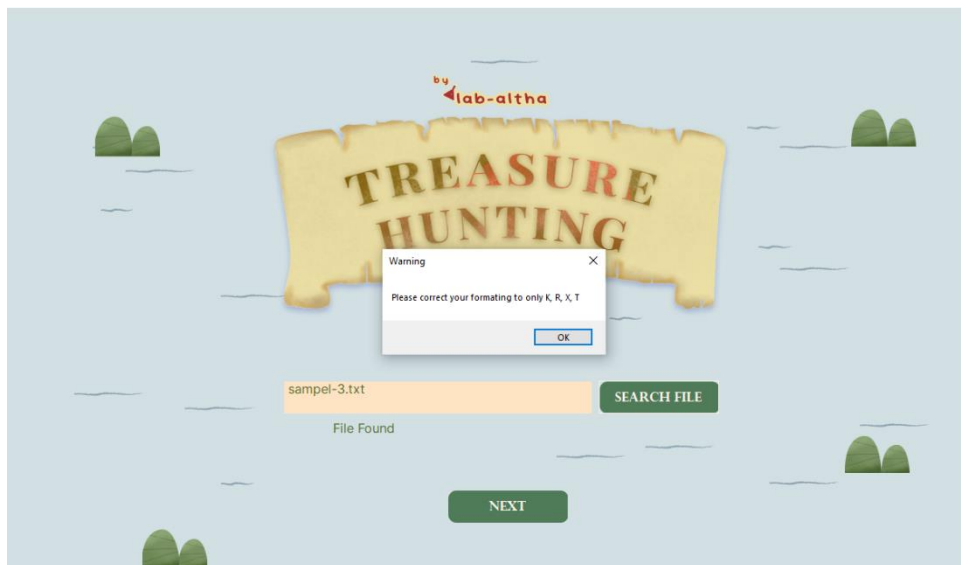


Gambar 4.4.2.5 Sampel-2 DFS with TSP

3. Sampel-3.txt

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

Gambar 4.4.3.1 Sampel-3.txt

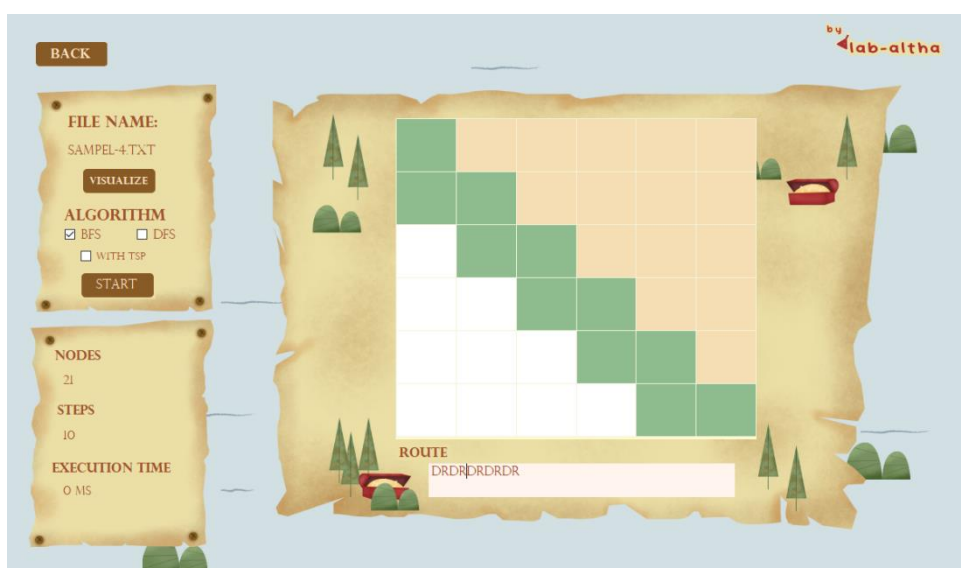


Gambar 4.4.3.2 Sampel-3

4. Sampel-4.txt

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

Gambar 4.4.4.1 Sampel-4.txt



Gambar 4.4.4.2 Sampel-4 BFS



Gambar 4.4.4.3 Sampel-4 BFS with TSP



by lab-alpha

BACK

FILE NAME:
SAMP1-4.TXT

VISUALIZE

ALGORITHM
☐ BFS ☒ DFS
☒ WITH TSP

START

NODES
33

STEPS
32

EXECUTION TIME
0 MS

ROUTE
DRDRDRDRLLLLLURRRRLULLLURRLULU

[illegible]

Gambar 4.4.5.1 Sampel-5.txt



Gambar 4.4.5.1 BFS



The screenshot shows a web application for solving a maze. The interface is divided into a sidebar on the left and a main workspace on the right.

Sidebar Controls:

- BACK** button.
- FILE NAME:** `SAMPEL-5.TXT`
- VISUALIZE** button.
- ALGORITHM:**
 - ☐ BFS
 - ☒ DFS
 - ☐ WITH TSP
- START** button.
- NODES:** 23
- STEPS:** 22
- EXECUTION TIME:** 0 MS

Main Workspace:

- A maze grid is displayed, with the solution path highlighted in red.
- The word **ROUTE** is shown below the grid.
- The route sequence is displayed in a text box: `RRDDDD|DDDDDDLUUUUUUUUUUU`.

The background of the workspace features a light blue sky and green hills, with a red car icon positioned near the maze.

BACK

FILE NAME:
SAMPLE.TXT

VISUALIZE

ALGORITHM

☐ BFS
☒ DFS

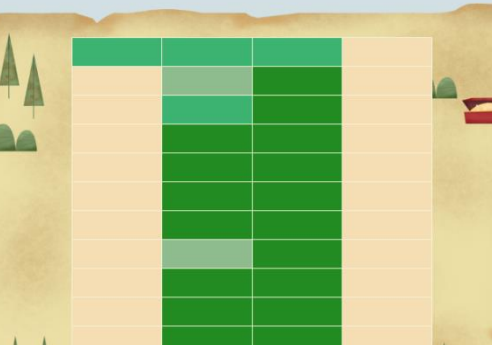
☒ WITH TSP

START

NODES
63


STEPS
62

EXECUTION TIME
0 MS



ROUTE

RRDDDDDDDDDLUUUUUUUUURDDDDDDDD
LUUUUUUUUDDDDDDDRUUUUUUUUUUUU

by  lab-alpha

4.5 Analisis Desain Solusi Algoritma BFS dan DFS

1. Berdasarkan hasil pengujian sampel-1 terlihat bahwa path dari BFS adalah sebagai berikut R-U-U-D-R-R-D. Sedangkan path dari DFS adalah sebagai berikut R-U-U-D-R-R-D-U-L-L-D-L. Dalam kasus ini terlihat bahwa BFS lebih efektif dalam segi ruang maupun waktu.

2. Berdasarkan hasil pengujian sampel-2 terlihat bahwa path dari BFS adalah sebagai berikut L-R-R-R. Sedangkan path dari DFS adalah sebagai berikut R-R-L-L-L. Namun BFS memiliki execution time yaitu 1 ms dan DFS 0 ms. Pada kasus ini terlihat bahwa DFS lebih efektif dari sisi waktu.
3. Berdasarkan hasil pengujian sampel-4 terlihat bahwa path dari BFS adalah sebagai berikut D-R-D-R-D-R-D-R-D-R. Sedangkan path dari DFS adalah sebagai berikut D-R-D-R-D-R-D-R-D-R. Terlihat bahwa BFS dan DFS memiliki solusi yang sama pada kasus ini, berikut juga dengan execution time yang sama.
4. Berdasarkan hasil pengujian sampel-5 terlihat bahwa path dari BFS adalah sebagai berikut R-D-D-D-D-D-D-D-D-D-D. Sedangkan path dari DFS adalah sebagai berikut R-R-D-D-D-D-D-D-D-D-D-D-L-U-U-U-U-U-U-U-U-U-U-U-U. Terlihat bahwa pada kasus ini juga BFS lebih efektif dari sisi ruang.

Berdasarkan kelima sampel uji di atas dapat disimpulkan bahwa DFS lebih efektif dari segi waktu dibandingkan dengan BFS dengan rata-rata waktu eksekusi mendekati 0 detik. Sedangkan pada beberapa kasus BFS lebih efektif dari sisi ruang penyelesaian, hal ini juga dapat terjadi karena prioritas arah yang telah diatur. Prioritas arah dapat memengaruhi algoritma DFS dalam proses pencariannya yang dapat menyebabkan pencarian mendalam jauh lebih luas. Dengan prioritas arah yang berbeda algoritma DFS mungkin dapat lebih efektif baik dalam segi waktu maupun ruang.

BAB V

Kesimpulan dan Saran

5.1 Kesimpulan

Dari tugas besar ini, kami telah berhasil memanfaatkan algoritma dengan pendekatan DFS dan BFS untuk melakukan pencarian solusi yaitu jalur untuk menemukan *treasure* pada *Maze Treasure Hunt*. Dalam tugas besar ini, kami mengembangkan algoritma dasar DFS dan BFS untuk dapat melakukan pencarian *treasure* pada *Maze Treasure Hunt*. Dalam pengaplikasiannya algoritma umum DFS dan BFS mendominasi algoritma kami sehingga hanya perlu melakukan modifikasi pada algoritma dasar tersebut seperti menambahkan posisi berhenti pencarian, mencatat jalur solusi, dll.

5.2 Saran

Saran-saran dalam pengerjaan Tugas Besar 2 IF2211 Strategi dan Algoritma Semester 2 Tahun Ajaran 2022/2023 adalah sebagai berikut:

1. Untuk para asisten dan tim pengajar, untuk membuat spesifikasi tugas lebih rinci dan jelas lagi. Selain itu, kami ingin memberikan saran kepada asisten supaya dapat menjawab QnA dengan lebih cepat.
2. Dalam pengerjaan tugas besar ini, perlu dilakukan *testing* dari beberapa sampel yang variatif sehingga kekurangan pada program lebih mudah untuk ditemukan dan strategi DFS dan BFS yang digunakan dapat lebih dikembangkan lagi.
3. Pengembangan aplikasi desktop dengan *framework* WinForms hanya dapat digunakan dan dikembangkan melalui sistem operasi Windows. Untuk kedepannya, pengembangan aplikasi desktop dapat dikembangkan menggunakan *framework* yang berbeda sehingga aplikasi *desktop* dapat digunakan dan dikembangkan pada sistem operasi yang lebih beragam.

5.3 Refleksi

Sebagai refleksi, kami ingin melakukan evaluasi pada diri sendiri agar kedepannya dapat membuat *timeline* pengerjaan yang lebih jelas dan rapi sehingga dapat menyelesaikan tugas besar ini jauh dari *deadline*. Kami juga harus dapat meningkatkan komunikasi antaranggota agar penyatuan bagian masing-masing dapat lebih mudah serta struktur kode yang dihasilkan lebih seragam dan konsisten. Kami berharap bahwa dengan melakukan refleksi ini, kami dapat terus meningkatkan kemampuan kami dalam mengerjakan tugas besar dan pengetahuan mengenai strategi algoritma *Breadth-First*

Search (BFS) dan *Depth-First Search* (DFS) untuk mencapai hasil yang lebih baik di masa depan.

5.4 Tanggapan

Dalam pengerjaan tugas besar ini, kami belajar untuk menggunakan sebuah *framework* dalam pengembangan aplikasi desktop menggunakan bahasa pemrograman c#. *Framework* yang kami gunakan adalah WinForm App yang dikombinasikan dengan c# menghasilkan sebuah aplikasi desktop dengan *modern interface*.

DAFTAR PUSTAKA

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

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

LINK REPOSITORY

Repository program dapat diakses melalui:

Github: https://github.com/BrigitaCarolina/Tubes2_lab-altha

LINK VIDEO

Video dapat diakses melalui:

Youtube: [IF2211 Tugas Besar 2 Strategi Algoritma BFS DFS Treasure](#)

