

Laporan Tugas Besar 2
IF2211 Strategi Algoritma
“Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan
Persoalan *Maze Treasure Hunt*”



Disusun oleh:
Kelompok 10 (C-Keris)

Anggota:

- Kevin John Wesley Hutabarat (13521042)
- Manuella Ivana Uli Sianipar (13521051)
- Moh. Aghna Maysan Abyan (13521076)

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

SEMESTER II TAHUN AJARAN 2022/2023

DAFTAR ISI

DAFTAR ISI	1
BAB I DESKRIPSI TUGAS	2
BAB II LANDASAN TEORI	3
2.1. Dasar Teori	3
2.1.1. Traversal Graf	3
2.1.2. Breadth First Search (BFS)	4
2.1.3. Properti BFS	5
2.1.4. Depth First Search (DFS)	6
2.1.5. Properti DFS	8
2.2. C# Desktop Application Development	8
BAB III PEMECAHAN MASALAH	9
3.1. Langkah Pemecahan Masalah	9
3.2. Proses Mapping Persoalan	9
3.3. Contoh Ilustrasi Kasus Lainnya	10
BAB IV ANALISIS PEMECAHAN MASALAH	11
4.1. Implementasi Program (pseudocode)	11
4.2. Penjelasan Struktur Data Program	14
4.2.1. Map	14
4.2.2. Point	14
4.2.3. Solution	14
4.3. Tata Cara Penggunaan Program	14
4.4. Hasil Pengujian	18
4.4.1. sampel-1.txt	18
4.4.2. sampel-2.txt	19
4.4.3. sampel-3.txt	20
4.4.4. sampel-4.txt	21
4.4.5. sampel-5.txt	22
4.5. Analisis Desain Solusi Algoritma BFS dan DFS	24
BAB V KESIMPULAN DAN SARAN	25
5.1. Kesimpulan	25
5.2. Saran	25
5.3. Refleksi	25
5.4. Tanggapan	25
DAFTAR PUSTAKA	27
LAMPIRAN	28

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

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 menghandle 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.

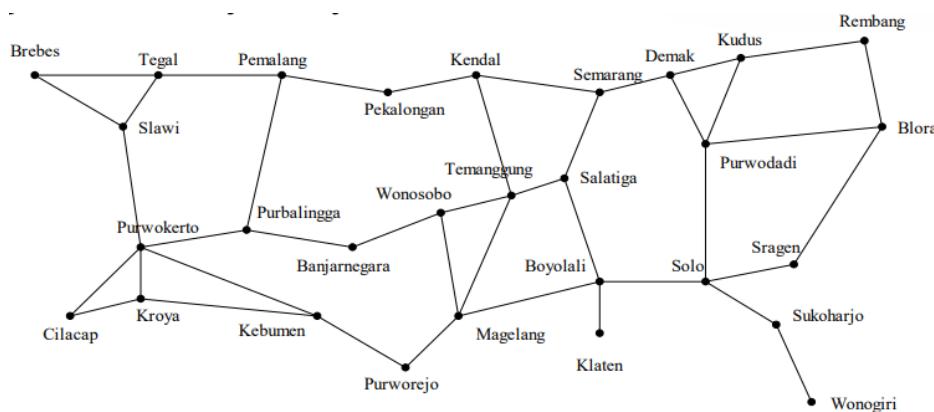
BAB II

LANDASAN TEORI

2.1. Dasar Teori

2.1.1. Traversal Graf

Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Graf memiliki 2 elemen utama, yaitu V dan E , dimana V (*vertices*) melambangkan himpunan tidak kosong dari simpul-simpul yang ada, dan E (*edges*) melambangkan himpunan sisi yang menghubungkan sepasang simpul.



Gambar 2.1 Pemetaan jaringan jalan raya yang menghubungkan sejumlah kota di Provinsi Jawa Tengah dengan menggunakan graf

Traversal graf merupakan algoritma yang aktivitasnya adalah mengunjungi simpul dengan cara yang sistematik, contohnya adalah *Breadth First Search* (BFS) dan *Depth First Search* (DFS) (asumsikan graf terhubung). Graf dapat dikatakan sebagai representasi persoalan dan traversal graf adalah pencarian solusinya.

Algoritma pencarian solusi ada 2, yaitu:

1. Tanpa informasi (uninformed/blind search)
 - Tidak ada informasi tambahan
 - Contoh: BFS, DFS, *Depth Limited Search*, *Iterative Deepening Search*, *Uniform Cost Search*
 2. Dengan informasi (informed search)
 - Pencarian berbasis heuristik, dengan mengetahui *non-goal state* yang “lebih menjanjikan” daripada yang lain
 - Contoh: *Best First Search*, A*

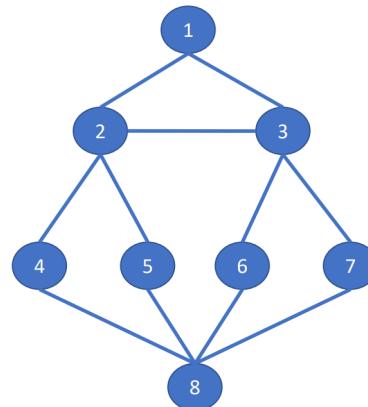
2.1.2. *Breadth First Search (BFS)*

Breadth First Search (BFS) merupakan salah satu contoh dari pendekatan graf statis. Sesuai namanya, BFS merupakan algoritma pencarian yang menelusuri sebuah graf secara melebar terlebih dahulu.

Misalkan traversal dimulai dari simpul v , maka algoritmanya adalah sebagai berikut:

1. Kunjungi simpul v .
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

Sebagai contoh, ambillah graf dengan bentuk sebagai berikut:



Gambar 2.2 Contoh Graf untuk BFS

Algoritma pencarian graf tersebut dengan metode BFS (dengan prioritas angka terkecil) adalah sebagai berikut:

1. Mulai pencarian dari simpul 1.

Urutan simpul sementara = {1}

Simpul yang belum dikunjungi = {2,3,4,5,6,7,8}

2. Kunjungi semua simpul yang terhubung dengan simpul 1, yaitu simpul 2 dan 3. Kunjungi simpul 2 terlebih dahulu karena $2 < 3$.

Urutan simpul sementara = {1,2,3}

Simpul yang belum dikunjungi = {4,5,6,7,8}

3. Semua simpul yang terhubung dengan simpul 1 telah dikunjungi. Mulailah mengunjungi semua simpul yang terhubung dengan simpul 2 dan belum

dikunjungi, yaitu simpul 4 dan 5. Kunjungi simpul 4 terlebih dahulu karena $4 < 5$.

Urutan simpul sementara = {1,2,3,4,5}

Simpul yang belum dikunjungi = {6,7,8}

4. Semua simpul yang terhubung dengan simpul 2 telah dikunjungi. Mulailah mengunjungi semua simpul yang terhubung dengan simpul 3 dan belum dikunjungi, yaitu simpul 6 dan 7. Kunjungi simpul 6 terlebih dahulu karena $6 < 7$.

Urutan simpul sementara = {1,2,3,4,5,6,7}

Simpul yang belum dikunjungi = {8}

5. Semua simpul yang terhubung dengan simpul 3 telah dikunjungi. Mulailah mengunjungi semua simpul yang terhubung dengan simpul 4 dan belum dikunjungi, yaitu simpul 8. Kunjungi simpul 8.

Urutan simpul sementara = {1,2,3,4,5,6,7,8}

Simpul yang belum dikunjungi = {}

6. Karena semua simpul sudah dikunjungi, proses pencarian selesai.

Urutan simpul = {1,2,3,4,5,6,7,8}

2.1.3. Properti BFS

Tabel 2.1 Properti BFS

Properti	Penjelasan Properti
<i>Completeness</i>	Ya (selama nilai b terbatas)
<i>Optimality</i>	Ya, jika langkah = biaya
Kompleksitas Waktu	$1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$
Kompleksitas Ruang	$O(b^d)$
Catatan tambahan	Kurang baik dalam kompleksitas ruang

Catatan:

- b : (*branching factor*) maksimum pencabangan yang mungkin dari suatu simpul
- d : (*depth*) kedalaman dari solusi terbaik (*cost* terendah)

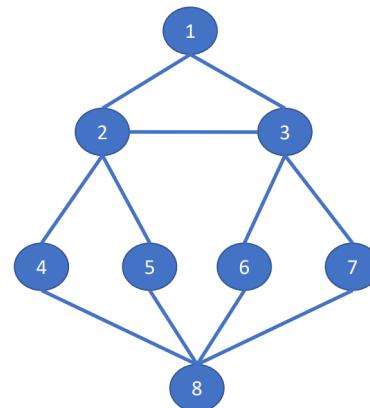
2.1.4. Depth First Search (DFS)

Depth First Search (DFS) merupakan contoh lainnya dari pendekatan graf statis. DFS merupakan algoritma pencarian yang menelusuri sebuah graf secara menurun/mendalam terlebih dahulu.

Misalkan traversal dimulai dari simpul v , maka algoritmanya adalah sebagai berikut:

1. Kunjungi simpul v .
2. Kunjungi simpul w yang bertetangga dengan simpul v .
3. Ulangi DFS mulai dari simpul w .
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian diruntut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.

Sebagai contoh, ambillah graf dengan bentuk sebagai berikut:



Gambar 2.3 Contoh Graf untuk DFS

Algoritma pencarian graf tersebut dengan metode BFS (dengan prioritas angka terkecil) adalah sebagai berikut:

1. Mulai pencarian dari simpul 1.

Urutan simpul sementara = {1}

Simpul yang belum dikunjungi = {2,3,4,5,6,7,8}

2. Kunjungi simpul prioritas yang terhubung dengan simpul 1 dan belum dikunjungi, yaitu simpul 2.

Urutan simpul sementara = {1,2}

Simpul yang belum dikunjungi = {3,4,5,6,7,8}

3. Kunjungi simpul prioritas yang terhubung dengan simpul 2 dan belum dikunjungi, yaitu simpul 3.

Urutan simpul sementara = {1,2,3}

Simpul yang belum dikunjungi = {4,5,6,7,8}

4. Kunjungi simpul prioritas yang terhubung dengan simpul 3 dan belum dikunjungi, yaitu simpul 6.

Urutan simpul sementara = {1,2,3,6}

Simpul yang belum dikunjungi = {4,5,7,8}

5. Kunjungi simpul prioritas yang terhubung dengan simpul 6 dan belum dikunjungi, yaitu simpul 8.

Urutan simpul sementara = {1,2,3,6,8}

Simpul yang belum dikunjungi = {4,5,7}

6. Kunjungi simpul prioritas yang terhubung dengan simpul 8 dan belum dikunjungi, yaitu simpul 4.

Urutan simpul sementara = {1,2,3,6,8,4}

Simpul yang belum dikunjungi = {5,7}

7. Kunjungi simpul prioritas yang terhubung dengan simpul 4 dan belum dikunjungi. Karena semua simpul yang terhubung dengan simpul 4 telah dikunjungi (2 dan 8), maka pencarian akan melakukan *backtrack* ke simpul sebelumnya, yaitu simpul 8. Saat sudah di simpul 8, kunjungi simpul prioritas yang terhubung dengan simpul 8 dan belum dikunjungi, yaitu simpul 5.

Urutan simpul sementara = {1,2,3,6,8,4,5}

Simpul yang belum dikunjungi = {7}

8. Kunjungi simpul prioritas yang terhubung dengan simpul 5 dan belum dikunjungi. Karena semua simpul yang terhubung dengan simpul 5 telah dikunjungi (2 dan 8), maka pencarian akan melakukan *backtrack* ke simpul sebelumnya, yaitu simpul 8. Saat sudah di simpul 8, kunjungi simpul prioritas yang terhubung dengan simpul 8 dan belum dikunjungi, yaitu simpul 7.

Urutan simpul sementara = {1,2,3,6,8,4,5,7}

Simpul yang belum dikunjungi = {}

9. Karena semua simpul sudah dikunjungi, proses pencarian selesai.

Urutan simpul = {1,2,3,6,8,4,5,7}

2.1.5. Properti DFS

Tabel 2.2 Properti DFS

Properti	Penjelasan Properti
<i>Completeness</i>	Ya (selama nilai b terbatas, dan ada penanganan ‘ <i>redundant paths</i> ’ dan ‘ <i>repeated states</i> ’)
<i>Optimality</i>	Tidak
Kompleksitas Waktu	$O(b^m)$
Kompleksitas Ruang	$O(bm)$
Catatan tambahan	Kurang baik dalam kompleksitas waktu, lebih baik dalam kompleksitas ruang

Catatan:

- b : (*branching factor*) maksimum pencabangan yang mungkin dari suatu simpul
- d : (*depth*) kedalaman dari solusi terbaik (*cost terendah*)
- m : maksimum kedalaman dari ruang status (bisa *infinite*)

2.2. C# Desktop Application Development

Desktop Application, atau dalam Bahasa Indonesia artinya aplikasi *desktop*, adalah program perangkat lunak yang berjalan secara lokal di dalam perangkat komputer. Aplikasi *desktop* tidak dapat diakses dari *browser*, seperti aplikasi berbasis jaringan (*web-based apps*) dan butuh penyebaran pada komputer pribadi atau laptop.

Bahasa C# merupakan bahasa pemrograman yang *general-purpose* dan berorientasi objek yang dibuat oleh Microsoft. Bahasa C# didesain sebagai bagian dari .NET *frameworks*, yang membuat bahasa C# dapat digunakan untuk pengembangan aplikasi *desktop*. Pengembangan aplikasi *desktop* pada C# dapat dilakukan pada IDE Visual Studio.

BAB III

PEMECAHAN MASALAH

3.1. Langkah Pemecahan Masalah

Langkah-langkah pemecahan masalah dari persoalan ini adalah sebagai berikut:

- 1) Mencari letak titik awal penelusuran pada peta.
- 2) Dari titik awal, penelusuran dilanjutkan pada titik yang bertetangga dengan titik awal.
- 3) Setiap kali sebuah titik diperiksa, titik-titik yang bertetangga dengan titik tersebut akan dimasukkan ke dalam antrian.
- 4) Setiap titik yang akan dimasukkan ke dalam antrian diperiksa dulu apakah sudah pernah ditelusuri atau tidak. Jika belum, maka titik akan dimasukkan ke dalam antrian.
- 5) Titik yang akan diperiksa diambil dari antrian.
- 6) Proses pemeriksaan akan berhenti jika semua *treasure* (harta karun) sudah ditemukan. Kemudian, program akan menampilkan rute ke *treasure* beserta arah-arah pergerakan dari awal sampai ke semua *treasure*.

3.2. Proses *Mapping* Persoalan

Proses *mapping* persoalan adalah sebagai berikut:

- 1) Program harus mencari semua *treasure* (harta karun) yang ada di map, artinya program harus mengetahui terlebih dahulu jumlah *treasure* yang ada di map.
- 2) Program hanya dapat bergerak ke atas, bawah, kiri, dan kanan dari posisinya.
- 3) Karena program tidak mengetahui informasi apapun mengenai titik-titik yang akan dituju, penelusuran harta karun dapat diselesaikan dengan algoritma *breadth first search* atau *depth first search*.
- 4) BFS menelusuri titik dengan level demi level, sedangkan DFS menelusuri titik dengan mengikuti terlebih dahulu suatu jalur sampai akhir. Oleh karena itu, struktur data yang tepat untuk digunakan pada antrian BFS adalah *queue*, sedangkan untuk DFS adalah *stack*.
- 5) Untuk setiap simpul yang diperiksa, simpul yang bertetangga dengan simpul tersebut akan dimasukkan ke dalam antrian dengan metode *enqueue* untuk BFS dan *push* untuk DFS.
- 6) Setiap kali simpul akan dimasukkan ke dalam antrian, harus diperiksa apakah simpul tersebut dapat dilalui atau tidak (K,T,R). Jika iya, maka simpul dapat dimasukkan ke dalam antrian.
- 7) Urutan prioritas pengecekan titik adalah atas, kanan, bawah, dan kiri.
- 8) Jika antrian kosong, maka list dari point yang sudah dicek dapat dikosongkan agar dapat dilalui kembali.

- 9) Struktur data titik (point) harus menyimpan list titik-titik yang sudah dilalui dan pergerakan yang sudah dilakukan agar memudahkan dalam penampilan rute solusi.

3.3. Contoh Ilustrasi Kasus Lainnya

Algoritma pencarian BFS dan DFS dapat diterapkan untuk kasus-kasus berikut:

- 1) *Citation Map*
- 2) *Web Spider*
- 3) *Folder Directory Search*
- 4) *Route/Path Finding*
- 5) Pembentukan Pohon Ruang Status
- 6) Permainan 8-Puzzle
- 7) *Maze Puzzle*
- 8) Dan lain-lainnya

BAB IV

ANALISIS PEMECAHAN MASALAH

4.1. Implementasi Program (*pseudocode*)

Berikut ini adalah pseudocode program untuk metode BFS:

```
procedure bfsSearch (input: Map m, output solution: Solution, listTreasure: array  
of Point, checkedPointCount: int)  
{I.S. Map yang valid, hanya berisi K, R, T, dan X}  
{F.S. Solusi pencarian treasure dengan metode breadth first search}
```

KAMUS LOKAL

```
antrian: Queue of Point  
map: Map  
checkedPoint: array of Point  
countTreasure, checkedPointCount: int  
listTreasure: array of (int,int)  
solution: Solution  
startPoint, point, up, right, down, left: (int,int)
```

ALGORITMA

```
startPoint ← startPoint(map) {Mengambil simpul start}  
addSearch(startPoint) {Menambahkan simpul start ke antrian}  
while (countTreasure != 0) do  
    {Mengambil satu simpul dari antrian dengan metode Dequeue}  
    point ← Dequeue(antrian)  
    checkedPointCount ← checkedPointCount + 1  
  
    {Jika ditemukan treasure, akan dimasukkan ke dalam listTreasure,  
    dan simpul akan dimasukkan ke dalam solution}  
    if (isTreasure(map,point) and not(alreadyFound(point)) then  
        countTreasure ← countTreasure - 1  
        addTreasure(point)  
        Clear(antrian)  
        addSolution(point)  
  
    {Menambahkan simpul yang bertetangga ke dalam antrian}  
    right ← getRight(map,point)  
    down ← getDown(map,point)  
    left ← getLeft(map,point)  
    up ← getUp(map,point)  
    addSearch(right)  
    addSearch(down)  
    addSearch(left)  
    addSearch(up)
```

```

{jika tidak ada lagi simpul yang dapat diperiksa, checkedPoint
dikosongkan terlebih dahulu}
if (Count(antrian) = 0) then
    emptyCheckedPoint()
    addSearch(right)
    addSearch(down)
    addSearch(left)
    addSearch(up)

{simpul yang sudah diperiksa dimasukkan ke dalam checkedPoint
insertChecked(point)}

```

procedure addSearch (input: point: (int,int), m: Map, input/output: antrian: Queue of Point)
{ I.S. Map yang valid, hanya berisi K,R,T dan X, point sembarang, antrian yang valid}
{ F.S. antrian yang sudah ditambahkan point jika memenuhi syarat}

KAMUS LOKAL

-

ALGORITMA

```

if( m!= -1 and n != -1) then
    if( isJalan(m, point) and not (alreadyChecked(point))) then
        Enqueue(antrian,point)

```

Berikut ini adalah pseudocode program untuk metode DFS:

procedure dfsSearch (input: Map m, output solution: Solution, listTreasure: array of Point, checkedPointCount: int)
{I.S. Map yang valid, hanya berisi K, R, T, dan X}
{F.S. Solusi pencarian treasure dengan metode depth first search}

KAMUS LOKAL

antrian: Stack of Point
map: Map
checkedPoint: array of Point
countTreasure, checkedPointCount: int
listTreasure: array of (int,int)
solution: Solution
startPoint, point, up, right, down, left: (int,int)

ALGORITMA

```

startPoint ← startPoint(map) {Mengambil simpul start}
addSearch(startPoint) {Menambahkan simpul start ke antrian}

```

```

while (countTreasure != 0) do:
    {Mengambil satu simpul dari antrian dengan metode Pop}
    point ← Pop(antrian)
    checkedPointCount ← checkedPointCount + 1

    {Jika ditemukan treasure, akan dimasukkan ke dalam listTreasure,
    dan simpul akan dimasukkan ke dalam solution}
    if (isTreasure(map,point) and not(alreadyFound(point)) then
        countTreasure ← countTreasure - 1
        addTreasure(point)
        Clear(antrian)
        addSolution(point)

    {Menambahkan simpul yang bertetangga ke dalam antrian}
    right ← getRight(map,point)
    down ← getDown(map,point)
    left ← getLeft(map,point)
    up ← getUp(map,point)
    addSearch(right)
    addSearch(down)
    addSearch(left)
    addSearch(up)

    {Jika tidak ada lagi simpul yang dapat diperiksa, checkedPoint
    dikosongkan terlebih dahulu}
    if (Count(antrian) = 0) then:
        emptyCheckedPoint()
        addSearch(right)
        addSearch(down)
        addSearch(left)
        addSearch(up)

    {simpul yang sudah diperiksa dimasukkan ke dalam checkedPoint
    insertChecked(point)}

```

procedure addSearch (input: point: (int,int), m: Map, input/output: antrian: Stack of Point)
{ I.S. Map yang valid, hanya berisi K,R,T dan X, point sembarang, antrian yang valid}
{ F.S. antrian yang sudah ditambahkan point jika memenuhi syarat}

KAMUS LOKAL

-

ALGORITMA

if(m!= -1 and n != -1) then

```
if( isJalan(m, point) and not (alreadyChecked(point))) then  
    Push(antrian,point)
```

4.2. Penjelasan Struktur Data Program

Struktur data yang digunakan dalam program adalah:

4.2.1. Map

Map merupakan sebuah kelas yang digunakan untuk merepresentasikan peta dari file yang *di-import* ke dalam bentuk matriks. Kelas map sendiri memiliki atribut m (jumlah baris dari peta), n (jumlah kolom dari peta) dan data (representasi peta dalam bentuk matriks/ array berdimensi dua).

4.2.2. Point

Point merupakan sebuah kelas yang digunakan untuk merepresentasikan simpul dalam peta. Setiap Point berisikan atribut berupa point (posisi simpul dalam matriks), listPreviousPoint (simpul apa saja yang sudah dilalui dari start sampai ke simpul tersebut), dan directions (kumpulan arah yang sudah dilalui dari simpul start ke simpul tersebut).

4.2.3. Solution

Solution merupakan sebuah kelas yang digunakan untuk menyimpan solusi dari BFS ataupun DFS. Solution memiliki atribut berupa listPoint (simpul-simpul yang menjadi rute solusi pencarian treasure) dan listDirection (arah-arah yang menjadi solusi, pergerakan dari simpul start ke treasure-treasure).

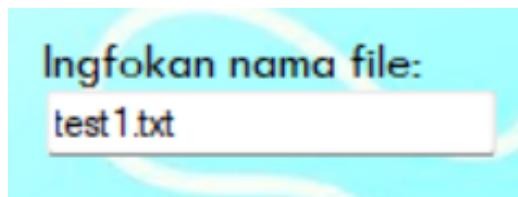
4.3. Tata Cara Penggunaan Program

Setelah mengikuti langkah-langkah meng-*compile* program pada README.md di repository GitHub kami, maka pengguna akan melihat tampilan layar seperti ini:



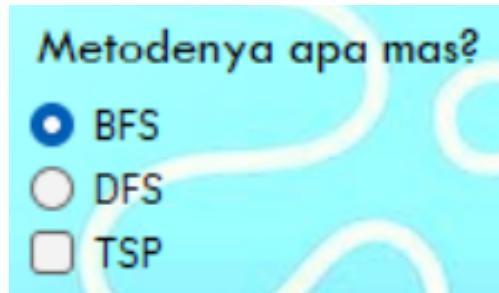
Gambar 4.1 Tampilan awal program

Pengguna perlu melakukan beberapa aksi sebelum program dapat memvisualisasikan hasil. Yang pertama, pengguna perlu memasukan nama *file* pada kotak input di bawah kalimat "Ingfokan nama file:". Semua *file* berisikan *map* yang ingin di tes terletak pada folder "test". Jika pengguna ingin menambahkan *file map* miliknya sendiri, silahkan buat *file* nya sendiri dalam bentuk .txt, lalu taruh *file* tersebut pada folder "test".

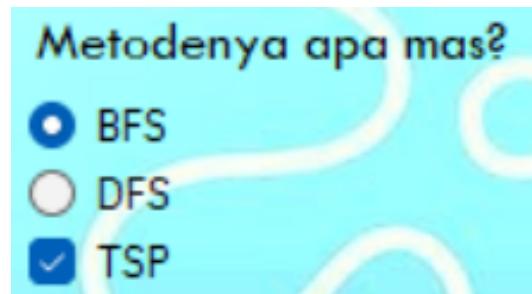


Gambar 4.2 Tampilan program saat pengguna memasukan nama *file*

Langkah kedua, pengguna dapat memilih metode yang diinginkan. Ada 2 pilihan utama, yaitu algoritma *Breadth First Search* (BFS) dan *Depth First Search* (DFS). Pengguna hanya dapat memilih salah satu pada saat ingin menjalankan visualisasi. Ada 1 pilihan tambahan, yaitu algoritma *Travelling Salesman Problem* (TSP). TSP bersifat opsional (dapat dipilih/tidak), namun jika ingin memakai TSP, pengguna harus tetap memilih salah satu antara BFS/DFS.



Gambar 4.3 Tampilan program jika pengguna hanya memilih BFS



Gambar 4.4 Tampilan program jika pengguna memilih TSP dengan BFS

Langkah selanjutnya bersifat opsional, yaitu menekan tombol “VISUALIZE”. Tombol ini berfungsi untuk memvisualisasikan peta berdasarkan *file* yang telah dimasukan pada langkah pertama. Pemilihan metode tidak berpengaruh karena visualisasi tidak ada kaitannya dengan metode yang dipilih.



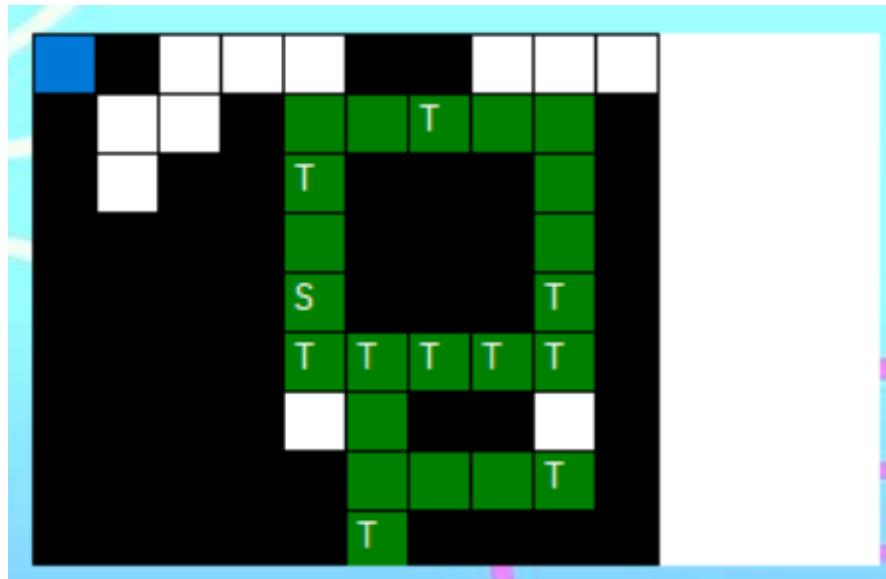
Gambar 4.5 Tampilan visualisasi *file*

Pada peta (Map), terdapat beberapa kotak dengan beda warna. Warna hitam menunjukkan rute yang tidak boleh dilalui (simbol X), warna putih menunjukkan rute yang boleh dilalui (simbol R). Untuk warna lainnya, terdapat keterangan yang menunjukkan makna dari tiap warna tersisa.



Gambar 4.6 Tampilan keterangan pada peta

Setelah pengguna melakukan semua masukan yang diperlukan, pengguna dapat menjalankan algoritma pencarian dengan menekan tombol “START”. Program akan mulai melaksanakan algoritma pencarian serta menampilkan solusinya.



Gambar 4.7 Tampilan peta setelah program selesai mencari rute

Selain itu, program juga menampilkan informasi lebih rinci dari hasil eksekusi yang telah dijalankan. Program menampilkan informasi penggunaan metode, rute yang diakses, jumlah node yang dilewati, langkah yang dibutuhkan ke kotak *Treasure*, dan waktu eksekusi program.

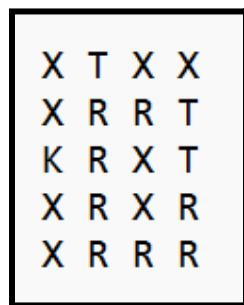
Using BFS
Using TSP: False
Route: D-R-R-R-R-U-U-U-L-L-L-L-D-D-D-D-R-D-D-D-D-L-R-U-U-R-R-R
Nodes: 67
Steps: 29
Execution time: 3.0057 ms

Gambar 4.8 Tampilan informasi rinci program setelah eksekusi

4.4. Hasil Pengujian

Kami mengambil data dari 5 file sampel yang dikirimkan dari spek, dengan 5 ukuran yang berbeda-beda.

4.4.1. sampel-1.txt



Gambar 4.9 Bentuk spesifikasi input file sampel-1.txt

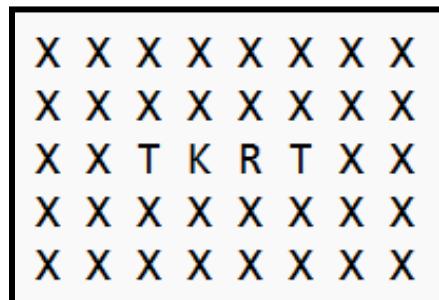


Gambar 4.10 Hasil pencarian harta karun pada sampel-1.txt dengan metode BFS



Gambar 4.11 Hasil pencarian harta karun pada sampel-1.txt dengan metode DFS

4.4.2. sampel-2.txt



Gambar 4.12 Bentuk spesifikasi input file sampel-1.txt

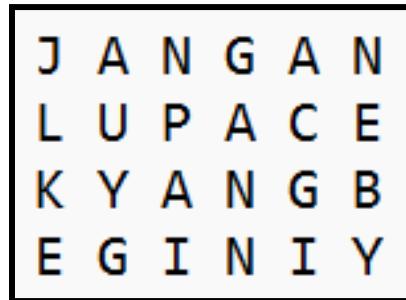


Gambar 4.13 Hasil pencarian harta karun pada sampel-2.txt dengan metode BFS



Gambar 4.14 Hasil pencarian harta karun pada sampel-2.txt dengan metode DFS

4.4.3. sampel-3.txt



Gambar 4.15 Bentuk spesifikasi input file sampel-3.txt



Gambar 4.16 Hasil visualisasi *file* sampel-3.txt. Peta tidak valid karena adanya komponen input selain K, T, R, dan X

4.4.4. sampel-4.txt

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

Gambar 4.17 Bentuk spesifikasi input *file* sampel-4.txt



Gambar 4.18 Hasil pencarian harta karun pada sampel-4.txt dengan metode BFS



Gambar 4.19 Hasil pencarian harta karun pada sampel-4.txt dengan metode DFS

4.4.5. sampel-5.txt

Gambar 4.20 Bentuk spesifikasi input file sampel-5.txt



Gambar 4.21 Hasil pencarian harta karun pada sampel-5.txt dengan metode BFS



Gambar 4.22 Hasil pencarian harta karun pada sampel-5.txt dengan metode DFS

4.5. Analisis Desain Solusi Algoritma BFS dan DFS

Keefektifan metode pencarian solusi algoritma BFS dan DFS sangat dipengaruhi oleh map. Pada beberapa kasus, BFS bisa lebih efisien dari DFS, sedangkan pada kasus lain DFS bisa lebih efisien dari BFS.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Algoritma *Breadth First Search* (BFS) dan *Depth First Search* (DFS) dapat digunakan untuk pemecahan solusi dalam permasalahan labirin. Penggunaan aplikasi Visual Studio pada tugas besar ini sangat berperan pada hasil akhir program, karena program dapat memvisualisasikan labirin dan rute yang dilalui oleh program.

Waktu *run* program dalam mencari solusi permasalahan dapat dikatakan cepat, bahkan beberapa *test case* menghasilkan waktu *run* yang sangat kecil sehingga program mencetak waktu 0 detik. Salah satu alasannya adalah karena C# merupakan bahasa yang dapat dikatakan cepat, bahkan lebih cepat jika dibandingkan dengan bahasa Python yang digunakan saat penggerjaan tugas kecil 2.

5.2. Saran

Dari tugas yang sudah dikerjakan, kelompok kami memiliki beberapa saran, saran yang kami miliki adalah sebagai berikut:

- 1) Sebelum implementasi algoritma, struktur data harus dipikirkan secara matang supaya tidak diperlukan perubahan masif selama membuat algoritma pencarian.
- 2) Diperlukan pemahaman lebih awal terkait Visual Studio dan bahasa C# supaya mahasiswa lebih siap menghadapi tugas besar ini.
- 3) Diperlukan perangkat/*device* dengan spesifikasi yang memadai untuk mengerjakan tugas besar, dikarenakan perlunya aplikasi Visual Studio yang memakan cukup banyak ruang.

5.3. Refleksi

Dalam penggerjaan tugas besar kali ini, kami merasa bahwa kami telah melakukan pekerjaan yang cukup baik dengan pembagian tugas yang cukup merata, meskipun kami baru bisa mulai fokus mengerjakan tugas besar ini pada hari Jum'at, 17 Maret 2023, 7 hari sebelum *deadline* tugas besar ini, dikarenakan kami sedang fokus mengerjakan tugas besar lainnya yang *deadline*-nya jatuh lebih awal. Kami juga merasa senang dengan hasil akhir serta dalam pembuatan program pada tugas besar ini, terutama dalam pembuatan GUI program.

5.4. Tanggapan

Tanggapan kami terkait tugas besar ini adalah, kami merasa terbebani di awal penggerjaan tugas besar ini dikarenakan kami belum mengetahui sintaks dan cara membuat program dengan bahasa C#, dengan kata lain, kami belum pernah memakai

bahasa C# sebelum turunnya tugas besar ini. Meskipun begitu, kami rasa beban tugas besar 2 lebih ringan jika dibandingkan dengan tugas besar 1.

DAFTAR PUSTAKA

- Munir, Rinaldi dan Nur Ulfa Maulidevi. 2021. *Breadth/Depth First Search (BFS/DFS) (Bagian 1)*. Homepage Rinaldi Munir. Diakses pada 20 Maret 2023 melalui <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>
- Munir, Rinaldi dan Nur Ulfa Maulidevi. 2021. *Breadth/Depth First Search (BFS/DFS) (Bagian 2)*. Homepage Rinaldi Munir. Diakses pada 20 Maret 2023 melalui <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>
- Munir, Rinaldi. 2022. *Graf (Bag.1)*. Homepage Rinaldi Munir. Diakses pada 20 Maret 2023 melalui <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>
- Rohn, Samantha. 2022. *What Is a Desktop Application? +Challenges, Use Cases*. Whatfix. Diakses pada 20 Maret 2023 melalui <https://whatfix.com/blog/desktop-application/>
- The Good and the Bad of C# Programming*. (29 Oct. 2021). AltexSoft. Diakses pada 20 Maret 2023 melalui <https://www.altexsoft.com/blog/c-sharp-pros-and-cons/>

LAMPIRAN

- Link repository GitHub:
https://github.com/AghnaAbyan/Tubes2_C-Keris
- Link YouTube:
<https://youtu.be/Np66b5vd0XI>