

Laporan Tugas Besar 2

IF2211 Strategi Algoritma

Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt



Kelompok krustycrew
Eunice Sarah Siregar / 13521013
M. Syauqi Jannatan / 13521014
Jauza Lathifah Annassalafi / 13521030

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Tahun 2022/2023

DAFTAR ISI

DAFTAR ISI	2
BAB I	3
DESKRIPSI TUGAS	3
BAB II	5
LANDASAN TEORI	5
2.1 Dasar Teori Graph Traversal, BFS, DFS Secara Umum	5
2.2 Penjelasan Singkat C# Desktop Application Development	6
BAB III	7
APLIKASI BFS DAN DFS	7
3.1 Langkah Pemecahan Masalah	7
3.2 Mapping menjadi BFS dan DFS	7
3.3 Contoh ilustrasi kasus lain	8
BAB IV	9
ANALISIS DAN PEMECAHAN MASALAH	9
4.1 Pseudocode Program Utama	9
4.2 Struktur Data dan Spesifikasi Program	16
4.3 Hasil Pengujian	17
4.4 Analisis Desain Solusi	18
BAB V	19
KESIMPULAN DAN SARAN	19
5.1 Kesimpulan dan Saran	19
5.2 Refleksi dan Tanggapan	19
DAFTAR PUSTAKA	20
LAMPIRAN	21

BAB I

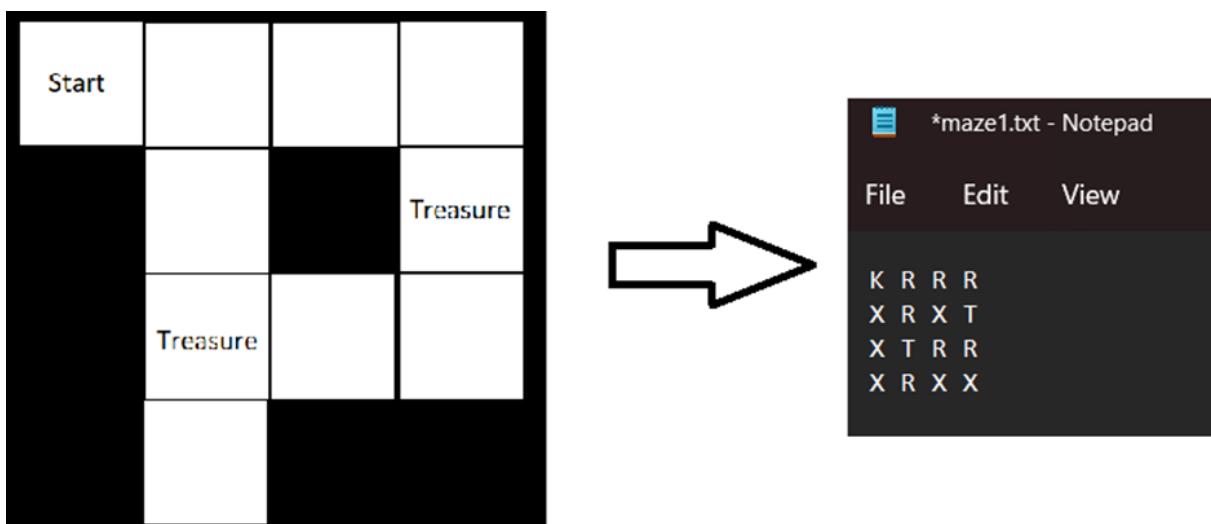
DESKRIPSI TUGAS



Gambar 1. Labirin di Bawah Krusty Krab

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 segiempat 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



Gambar 2. Ilustrasi input file maze

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 meng-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.

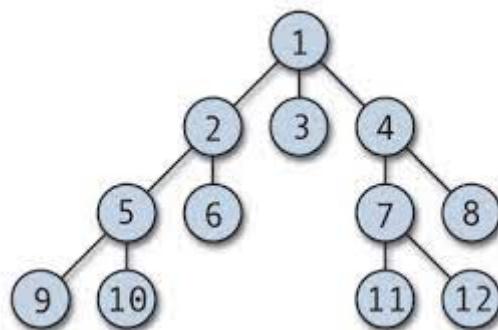
BAB II

LANDASAN TEORI

2.1 Dasar Teori Graph Traversal, BFS, DFS Secara Umum

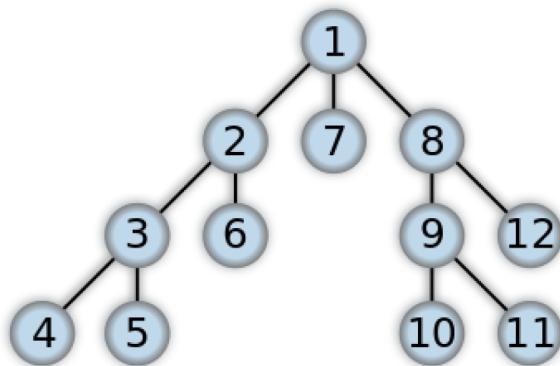
Graph traversal adalah proses mengunjungi setiap simpul atau *vertex* yang ada pada sebuah graph atau jaringan dengan mengikuti beberapa aturan tertentu. Pada dasarnya, terdapat dua metode utama dalam graf *traversal* yaitu Breadth-First Search (BFS) dan Depth-First Search (DFS).

BFS adalah metode traversal pada sebuah graph dimulai dari simpul awal, kemudian mengunjungi semua simpul yang terhubung secara bertahap, dan menghindari mengunjungi simpul yang sudah dikunjungi sebelumnya. Pada BFS, algoritma mengunjungi semua simpul pada tingkat yang sama terlebih dahulu sebelum bergerak ke tingkat berikutnya. Simpul-simpul yang telah dikunjungi akan dimasukkan ke dalam suatu antrean dan akan dilakukan pengecekan apakah simpul awal tersebut merupakan solusi atau tidak. Berikut adalah ilustrasi pencarian menggunakan algoritma BFS.



Gambar 3. Ilustrasi BFS

DFS adalah metode traversal pada sebuah graf dimulai dari simpul awal dan mengunjungi simpul terdekat yang belum dikunjungi sebelumnya, kemudian terus bergerak ke simpul berikutnya hingga mencapai simpul yang tidak dapat dikunjungi lagi. Pada DFS, algoritma akan mengunjungi semua simpul pada cabang yang sama terlebih dahulu sebelum beralih ke cabang lain. Simpul yang dikunjungi akan dimasukkan ke dalam tumpukan dan dilakukan pengecekan setiap tumpukan teratas. Berikut adalah ilustrasi pencarian menggunakan algoritma DFS.



Gambar 4. Ilustrasi DFS

2.2 Penjelasan Singkat C# Desktop Application Development

C# Desktop Application Development adalah salah satu teknologi yang digunakan untuk membuat aplikasi desktop menggunakan bahasa pemrograman C#. C# adalah bahasa pemrograman yang dikembangkan oleh Microsoft dan dirilis pertama kali pada tahun 2000.

Pengembangan aplikasi desktop dengan C# dilakukan menggunakan aplikasi Visual Studio. Visual Studio menyediakan berbagai macam tools dan fitur yang sangat membantu dalam pengembangan aplikasi desktop seperti form designer, drag-and-drop UI elements, code editor, debugging tools, dan lain sebagainya.

Dalam pengembangan aplikasi desktop dengan C#, kita dapat menggunakan Windows Forms atau WPF (Windows Presentation Foundation) sebagai teknologi tampilan (UI). Windows Forms adalah teknologi tampilan yang lebih tua dan simpel, sedangkan WPF adalah teknologi tampilan yang lebih modern dan fleksibel.

C# Desktop Application Development juga memiliki kemampuan untuk mengintegrasikan aplikasi desktop dengan teknologi lain seperti database, web service, dan sebagainya. Hal ini memungkinkan pengembangan aplikasi desktop yang lebih kompleks dan terintegrasi dengan sistem lain.

BAB III

APLIKASI BFS DAN DFS

3.1 Langkah Pemecahan Masalah

- Dalam mengumpulkan semua treasure di map, diperlukan beberapa langkah.
1. Program akan membaca file .txt menggunakan dan mengubahnya ke dalam sebuah matriks.
 2. Matriks akan diubah menjadi sebuah matriks baru yang memiliki border 'X' disekelilingnya. Tujuannya adalah apabila node paling kanan diperiksa tidak terjadi eror karena terdapat node disebelah kanannya yang dapat diperiksa dan tidak akan menjadi kandidat jalur yang akan dilalui karena berupa 'X'.
 3. Akan dicari titik awal dan banyaknya treasure dalam sebuah labirin.
 4. Untuk mencari jalan menuju suatu titik ke titik lain akan digunakan fungsi DFSMaze dengan algoritma DFS dan BFSMaze dengan algoritma BFS. Secara garis besar, fungsi tersebut akan membuat stack/queue untuk menyimpan node yang akan diperiksa dan melakukan looping hingga stack/queue kosong yang didalamnya akan melakukan proses pop sebuah stack atau *dequeue* sebuah queue yang akan dimasukan kedalam sebuah variabel dan diperiksa tetangga kanan, bawah, atas, kiri dari node jika tetangganya dapat dilewati akan dimasukkan kedalam stack atau queue dan sebuah struktur data dictionary yang berisi key (diisi oleh node/tetangga yang dapat dilewati) dan value (diisi oleh node yang diperiksa tetangganya). Apabila variabel yang diperiksa merupakan node 'T' maka program akan berhenti. Lalu akan dilakukan penghapusan node yang hanya diperiksa dan tidak menjadi jalur yang dilewati.
 5. Selanjutnya akan dicari semua node yang berisi 'T' dengan melakukan looping sebanyak banyaknya 'T' dalam labirin yang didalam pengulangan tersebut akan dipanggil fungsi DFSMaze atau BFSMaze yang tiap hasilnya akan dimasukkan ke dalam sebuah list yang merupakan hasil akhir berupa jalur untuk mengunjungi semua 'T' yang ada di dalam labirin.

3.2 Mapping menjadi BFS dan DFS

Algoritma *breadth first search* diimplementasikan menggunakan konsep pencarian solusi dari graf dinamis. Dalam hal ini, dibutuhkan mengecek ketetanggaan dari titik awal tersebut. Titik tetangga tersebut akan dimasukkan ke dalam suatu *queue* sampai semua treasure ditemukan.

Algoritma *depth first search* juga diimplementasikan menggunakan konsep graf dinamis. Elemen tumpukan berisi simpul yang akan dikunjungi. Untuk

merepresentasikan graf, dapat dilakukan menggunakan ketetanggaan dari titik-titik tersebut.

3.3 Contoh ilustrasi kasus lain

Selain algoritma BFS dan DFS terdapat terdapat algoritma lain yang dapat mencari harta karun di dalam sebuah labirin, yaitu Algoritma IDS (Iterative Deepening Search). Secara teori, algoritma IDS dapat digunakan untuk mencari harta karun dalam labirin, karena pada dasarnya IDS melakukan pencarian secara berulang pada kedalaman yang semakin dalam. Namun, dalam praktiknya, IDS mungkin tidak seefektif DFS atau BFS dalam mencari jalur terpendek untuk mencari harta karun dalam labirin. Hal ini disebabkan karena IDS melakukan banyak duplikasi dan perhitungan ulang pada setiap iterasi yang mungkin membuatnya lebih lambat dan memakan waktu lebih lama dibandingkan DFS atau BFS.

Selain itu, IDS juga dapat menghabiskan memori yang lebih banyak, terutama ketika mencari jalur yang panjang atau rumit. Oleh karena itu, IDS mungkin bukan pilihan terbaik untuk mencari harta karun dalam labirin, algoritma ini mungkin tidak seefektif atau efisien seperti DFS atau BFS dalam praktiknya.

BAB IV

ANALISIS DAN PEMECAHAN MASALAH

4.1 Pseudocode Program Utama

```
function mazeWithBorder(Matrix)
    CREATE temp as new two-dimensional char array with dimensions
    [Matrix.GetLength(0) + 2, Matrix.GetLength(1) + 2]
    for i = 1 to Matrix.GetLength(0)
        for i = 1 to Matrix.GetLength(1)
            temp[i, j] = Matrix[i - 1, j - 1]
        endfor
    endfor

    nRow = Matrix.GetLength(0)
    nCol = Matrix.GetLength(1)
    for i = 0 to nRow + 1
        temp[i, 0] = 'X'
        temp[i, nCol + 1] = 'X'
    endfor

    for i = 0 to nCol+ 1
        temp[0, i] = 'X'
        temp[nRow + 1, i] = 'X'
    endfor

    for i = 0 to nRow + 1
        for j = 0 to nCol + 1
            PRINT temp[i, j]
        endfor
        PRINT newline
    endfor

    RETURN temp
endfunction

function findStart(Matrix)
    CREATE start as new Tuple
    nRow = Matrix.GetLength(0)
    nCol = Matrix.GetLength(1)
    for i = 0 to nRow
        for j = 0 to nCol
            if Matrix[i, j] = 'K'
                CREATE start as new Tuple(i, j)
                RETURN start
            endif
        endfor
    endfor
endfunction
```

IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS
Tahun 2022/2023

```
    RETURN start
    endfunction

function CountTreasure(Matrix)
    nRow = Matrix.GetLength(0)
    nCol = Matrix.GetLength(1)
    count = 0
    for i = 0 to nRow
        for j = 0 to nCol
            if Matrix[i, j] = 'T'
                count++
            endif
        endfor
    endfor
    RETURN count
    endfunction

function findNeighbor(current, Matrix)
    CREATE neighbor as list of char
    if Matrix[current.Item1, current.Item2 + 1] = 'R' or
Matrix[current.Item1, current.Item2 + 1] = 'T' then
        ADD 'R' to neighbor
    else
        ADD '' to neighbor
    endif
    if Matrix[current.Item1 + 1, current.Item2] ='R' or
Matrix[current.Item1 + 1, current.Item2] ='T' then
        ADD 'D' to neighbor
    else
        ADD '' to neighbor
    endif
    if Matrix[current.Item1 - 1, current.Item2] = 'R' or
Matrix[current.Item1 - 1, current.Item2] = 'T' then
        ADD 'U' to neighbor
    else
        ADD '' to neighbor
    endif
    if Matrix[current.Item1, current.Item2 - 1] = 'R' or
Matrix[current.Item1, current.Item2 - 1] = 'T' then
        ADD 'L' to neighbor
    else
        ADD '' to neighbor
    endif
    return neighbor
    endfunction

function DFSMaze(Matrix, start)
    PUSH start to stack
    ADD start as key to tempPath with value of Tuple of two integers
    (-1, -1)
```

IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS
Tahun 2022/2023

```
WHILE stack is not empty DO
    SET current as top element of stack and POP stack
    if current is not in nodeDFS then
        ADD current to nodeDFS
    endif
    if Matrix[current.Item1, current.Item2] is 'T' then
        break loop
    endif
    SET RDUL as the list of neighbors of current obtained from
function findNeighbor(current, Matrix)
    FOR EACH item in RDUL DO
        IF item is 'R' THEN
            SET neighbor as Tuple of two integers (current.Item1,
current.Item2 + 1)
            IF neighbor is not in tempPath THEN
                PUSH neighbor to stack
                ADD neighbor as key to tempPath with value of
current
            ENDIF
        ELSE IF item is 'D' THEN
            SET neighbor as Tuple of two integers (current.Item1 +
1, current.Item2)
            IF neighbor is not in tempPath THEN
                PUSH neighbor to stack
                ADD neighbor as key to tempPath with value of
current
            ENDIF
        ELSE IF item is 'U' THEN
            SET neighbor as Tuple of two integers (current.Item1 -
1, current.Item2)
            IF neighbor is not in tempPath THEN
                PUSH neighbor to stack
                ADD neighbor as key to tempPath with value of
current
            ENDIF
        ELSE IF item is 'L' THEN
            SET neighbor as Tuple of two integers (current.Item1,
current.Item2 - 1)
            IF neighbor is not in tempPath THEN
                PUSH neighbor to stack
                ADD neighbor as key to tempPath with value of
current
            ENDIF
        ENDIF
    ENDFOR
ENDWHILE

SET k as the index of the last element in tempPath
WHILE Matrix[tempPath.ElementAt(k).Key.Item1,
```

IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS
Tahun 2022/2023

```
tempPath.ElementAt(k).Key.Item2] is not 'T' DO
    REMOVE tempPath.ElementAt(k).Key from tempPath
    k--
ENDWHILE

RETURN deleteBacktrack(tempPath, Matrix)

function BFS(Matrix, start)
    Enqueue start to stack
    ADD start as key to tempPath with value of Tuple of two integers
(-1, -1)

    WHILE stack is not empty DO
        SET current as top element of stack and POP stack
        if current is not in nodeDFS then
            Enqueue current to nodeDFS
        endif
        if Matrix[current.Item1, current.Item2] is 'T' then
            break loop
        endif
        SET RDUL as the list of neighbors of current obtained from
function findNeighbor(current, Matrix)
    FOR EACH item in RDUL DO
        IF item is 'R' THEN
            SET neighbor as Tuple of two integers (current.Item1,
current.Item2 + 1)
            IF neighbor is not in tempPath THEN
                Enqueue neighbor to queue
                ADD neighbor as key to tempPath with value of
current
            ENDIF
            ELSE IF item is 'D' THEN
                SET neighbor as Tuple of two integers (current.Item1 +
1, current.Item2)
                IF neighbor is not in tempPath THEN
                    Enqueue neighbor to queue
                    ADD neighbor as key to tempPath with value of
current
                ENDIF
                ELSE IF item is 'U' THEN
                    SET neighbor as Tuple of two integers (current.Item1 -
1, current.Item2)
                    IF neighbor is not in tempPath THEN
                        Enqueue neighbor to queue
                        ADD neighbor as key to tempPath with value of
current
                    ENDIF
                    ELSE IF item is 'L' THEN
                        SET neighbor as Tuple of two integers (current.Item1,
```

IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS
Tahun 2022/2023

```
current.Item2 - 1)
    IF neighbor is not in tempPath THEN
        Enqueue neighbor to queue
        ADD neighbor as key to tempPath with value of
current
    ENDIF
ENDIF
ENDFOR
ENDWHILE

SET k as the index of the last element in tempPath
WHILE Matrix[tempPath.ElementAt(k).Key.Item1,
tempPath.ElementAt(k).Key.Item2] is not 'T' DO
    REMOVE tempPath.ElementAt(k).Key from tempPath
    k--
ENDWHILE

RETURN deleteBacktrack(tempPath, Matrix)

function deleteBacktrack(path, path, Matrix)
    CREATE newPath IDictionary<Tuple<int, int>, Tuple<int, int>> AS
new Dictionary<Tuple<int, int>, Tuple<int, int>>()
    CREATE current AS path.ElementAt(path.Count - 1).Value
    newPath.Add(path.ElementAt(path.Count - 1).Key,
path.ElementAt(path.Count - 1).Value)
    i = path.Count - 1
    WHILE current != path.ElementAt(0).Value DO
        IF current = path.ElementAt(i - 1).Key THEN
            current = path.ElementAt(i - 1).Value
            newPath.Add(path.ElementAt(i - 1).Key, path.ElementAt(i -
1).Value)
        ENDIF
        i--
    END WHILE
    CREATE realPath AS new Dictionary<Tuple<int, int>, Tuple<int,
int>>()
    FOR j = newPath.Count - 1 DOWNTO 0 DO
        realPath.Add(newPath.ElementAt(j).Key,
newPath.ElementAt(j).Value)
    ENDFOR
    RETURN realPath
ENDFUNCTION

function GetAllTrasureDFS(Matrix, treasure, start):
    path <- empty list of dictionary
    FinalPath <- empty list of tuple
    tempPath <- empty dictionary
    tempStart <- Tuple(0, 0)
    tempNextStart <- Tuple(0, 0)
```

IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS
Tahun 2022/2023

```
if treasure == 1:
    tempPath <- DFSMaze(Matrix, start)
    add tempPath to path
else:
    for i in range(0, treasure):
        tempPath <- DFSMaze(Matrix, start)
        origin <- first key of tempPath
        Matrix[origin.Item1, origin.Item2] <- 'R'
        start <- last key of tempPath
        Matrix[start.Item1, start.Item2] <- 'R'
        if i > 0:
            remove first key-value pair of tempPath
        add tempPath to path

for item in path:
    for item2 in item:
        add item2's key to FinalPath

return FinalPath

function GetAllTrasureBFS(Matrix, treasure, start):
    path <- empty list of dictionary
    FinalPath <- empty list of tuple
    tempPath <- empty dictionary
    tempStart <- Tuple(0, 0)
    tempNextStart <- Tuple(0, 0)

    if treasure == 1 then
        tempPath <- BFSMaze(Matrix, start)
        add tempPath to path
    else
        for i in range(0, treasure):
            tempPath <- BFSMaze(Matrix, start)
            origin <- first key of tempPath
            Matrix[origin.Item1, origin.Item2] <- 'R'
            start <- last key of tempPath
            Matrix[start.Item1, start.Item2] <- 'R'
            if i > 0:
                remove first key-value pair of tempPath
            add tempPath to path
    endif
    for item in path
        for item2 in item
            add item2's key to FinalPath
        endfor
    endfor
    return FinalPath
endfunction
```

IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS
Tahun 2022/2023

```
function Direction(path):
    direction <- empty string
    str <- 0
    for i in range(0, length of path - 1):
        if path[i].Item1 == path[i+1].Item1:
            if path[i].Item2 < path[i+1].Item2 then
                direction += "R"
            else
                direction += "L"
            endif
        else
            if path[i].Item1 < path[i+1].Item1 then
                direction += "D"
            else
                direction += "U"
            endif
        endif
        if str < length of path - 2 then
            direction += " - "
        else
            direction += ""
        endif
        str += 1
    return direction
endfunction

function printPathMaze(index, start, Matrix)
    FOR i FROM 1 TO Matrix.GetLength(0) - 1 DO
        FOR j FROM 1 TO Matrix.GetLength(1) - 1 DO
            IF Matrix[i, j] = 'R' THEN
                SET pathMaze[i, j] = "R"
            ENDIF
        ENDFOR
    ENDFOR

    FOR i FROM 0 TO index.Count - 1 DO
        SET pathMaze[index[i].Item1, index[i].Item2] = "O"
    ENDFOR

    SET pathMaze[start.Item1, start.Item2] = "K"

    FOR i FROM 1 TO Matrix.GetLength(0) - 1 DO
        FOR j FROM 1 TO Matrix.GetLength(1) - 1 DO
            PRINT pathMaze[i, j] + " "
        END FOR
        PRINT newline
    END FOR
ENDFUNCTION
```

4.2 Struktur Data dan Spesifikasi Program

Dalam pembuatan program, digunakan struktur data sebagai berikut:

1. Queue

Untuk mengimplementasikan algoritma BFS, digunakan struktur data Queue pada kelas System.Collections.Generic. Queue digunakan untuk menyimpan titik yang sudah dijalani oleh program saat mencari treasure dengan metode BFS (*breadth first search*). Berikut adalah implementasi dari *method* yang digunakan.

- Enqueue, digunakan untuk memasukkan titik ke dalam antrean
- Dequeue, digunakan untuk mengeluarkan titik dari antrean
- ElementAt, digunakan untuk menemukan elemen ke- dalam sebuah antrean

2. Stack

Algoritma DFS sangat dekat dengan struktur data stack pada kelas System.Collections.Generic. Sama seperti queue, stack digunakan untuk menyimpan titik yang sudah dijalani oleh program. Dengan prinsip *last in first out*, stack akan mengimplementasikan DFS dalam pencarian treasure. Berikut adalah implementasi dari method yang digunakan.

- Push, digunakan untuk menambahkan titik ke dalam tumpukan
- Pop, digunakan untuk menghapus titik dari dalam tumpukan
- Count, untuk mendapatkan jumlah elemen yang terkandung dalam stack

3. List

List atau yang sering disebut array, dapat menyimpan kumpulan dari data dengan tipe data yang sama. Pada C# dapat mendeklarasikan list dengan berbagai dimensi. Berikut adalah implementasi dari method yang digunakan.

- Add, digunakan untuk menambah elemen dari list
- Clear, menghapus keseluruhan elemen dari list
- Remove, menghapus salah satu elemen dari list
- Length, untuk mendapatkan jumlah elemen yang terkandung dalam list

4. Tuple

Tuple merupakan kumpulan dari data dengan tipe data yang berbeda. Berikut adalah implementasi dari method yang digunakan.

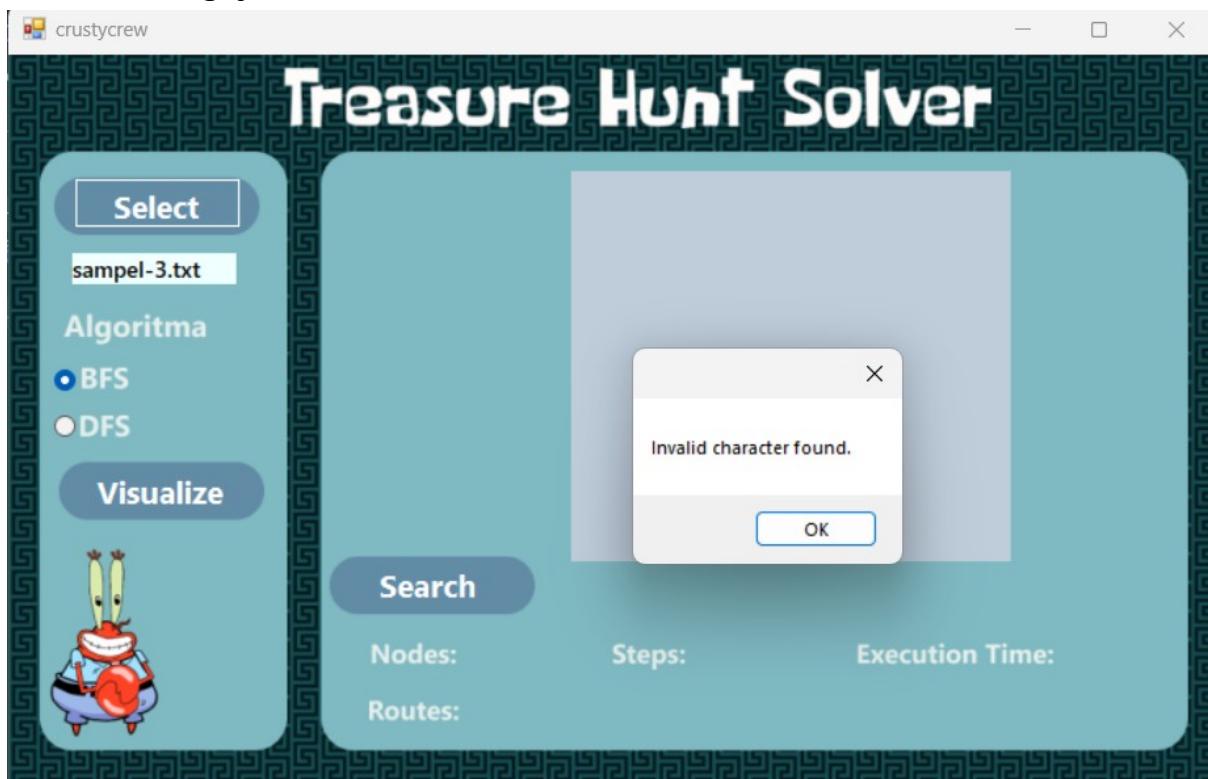
- Create, untuk menginisiasi awal pembentukan tuple

5. Dictionary

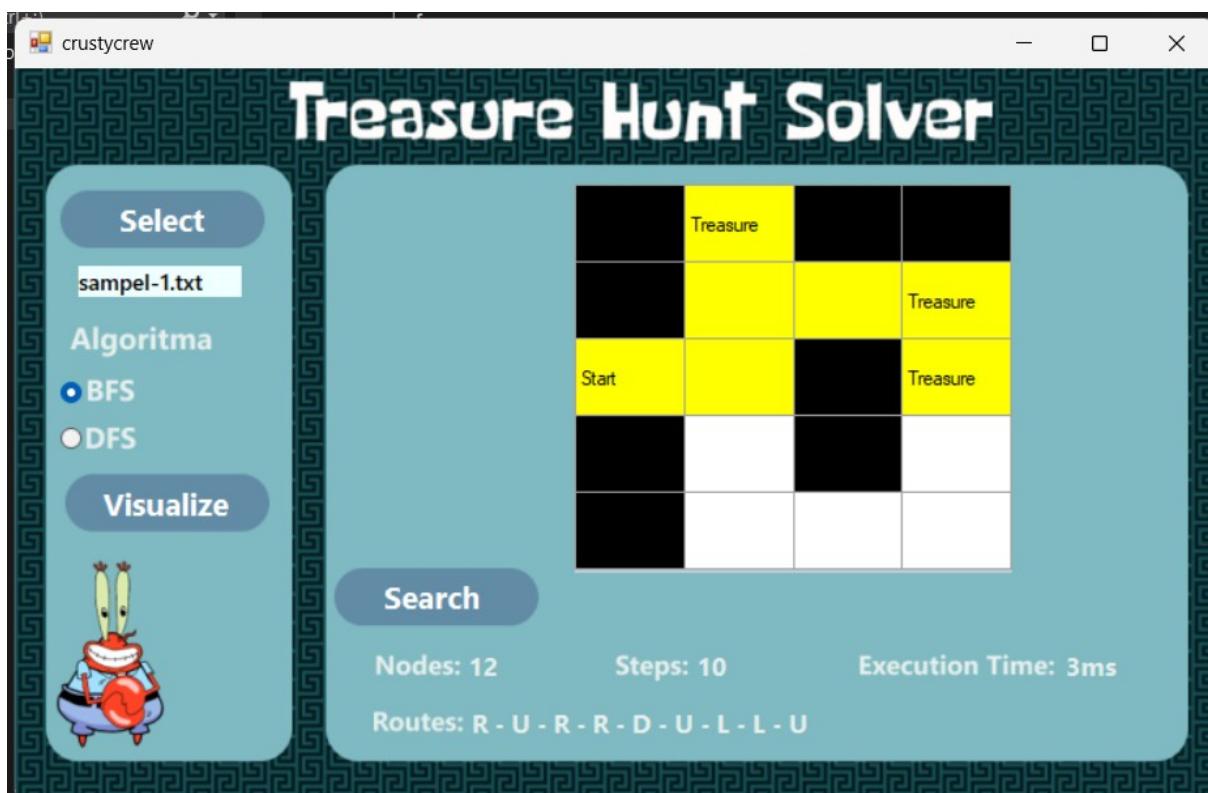
Dictionary adalah generic collection yang menyimpan pasangan key value tidak dengan particular order. Berikut adalah implementasi dari method yang digunakan.

- Add, digunakan untuk menambah elemen
- ElementAt, digunakan untuk menemukan elemen ke-

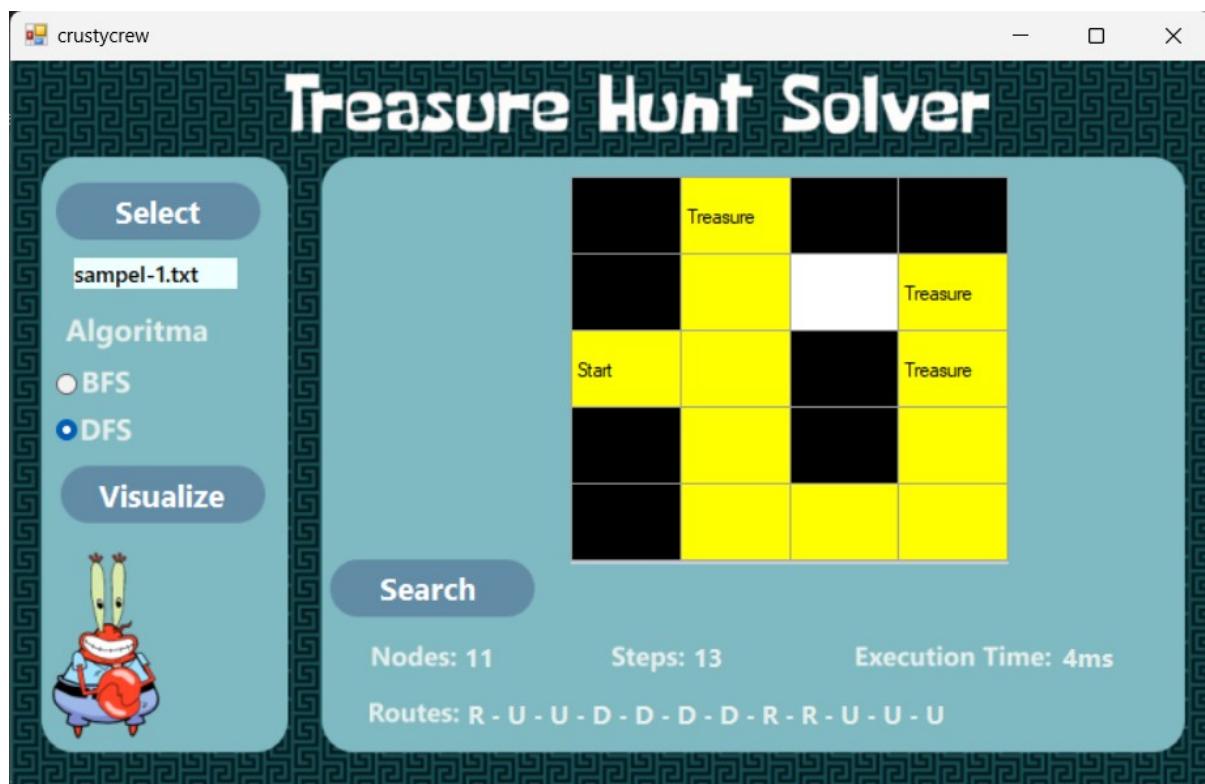
4.3 Hasil Pengujian



Gambar 5. Hasil pengujian ketika .txt memiliki nilai selain KRTX



Gambar 6. Hasil pengujian BFS



Gambar 7. Hasil pengujian DFS

4.4 Analisis Desain Solusi

Dalam pencarian harta karun di dalam labirin Algoritma BFS dan DFS mampu mendapatkan semua harta karun di dalam sebuah labirin yang tervalidasi. Algoritma DFS dapat menemukan harta karun dengan lebih baik/cepat jika harta karun tersebut berada di jarak yang dekat dengan titik awal dan memori yang dibutuhkan lebih sedikit karena hanya menyimpan satu jalur yang telah diproses. Namun, algoritma DFS akan mengalami sedikit halangan jika mencari harta karun yang letaknya jauh dari titik awal karena algoritmanya yang cenderung mengeksplorasi satu jalur secara penuh sebelum mencari jalur lainnya. Hal ini akan menghabiskan banyak waktu untuk menjelajahi area yang tidak mengandung harta karun, sehingga membutuhkan waktu yang lebih lama untuk menemukan harta karun. Sedangkan untuk algoritma BFS, akan menemukan harta karun dengan waktu yang lebih cepat ketika harta karun berada jauh dari titik awal dan akan menemukan harta karun terpendek dulu karena akan mengeksplorasi area terdekat terlebih dahulu, kemudian area yang lebih jauh. Namun, algoritma BFS akan membutuhkan memori lebih banyak karena menyimpan semua area yang mungkin untuk dijelajahi.

Secara keseluruhan, DFS akan lebih cepat dan membutuhkan memori lebih sedikit daripada BFS ketika harta karun berada dalam jarak dekat dengan titik awal. Sedangkan BFS lebih unggul ketika harta karun berada jauh dari titik awal dan memastikan menemukan harta karun terpendek.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan dan Saran

Algoritma *breadth-first search* dan *depth-first search* dapat digunakan dalam permasalahan pencarian jalan keluar dalam suatu labirin. BFS melakukan penelusuran secara horizontal terlebih dahulu dan akan disimpan ke dalam sebuah antrean, sedangkan DFS melakukan penelusuran secara vertikal terlebih dahulu dan akan disimpan ke dalam sebuah tumpukan. Untuk menjalankan program ini, pastikan memiliki .NET Framework 4.7.2 dan bahasa pemrograman C#.

5.2 Refleksi dan Tanggapan

Setelah menyelesaikan tugas besar ini, dapat direfleksikan bahwa pandai membagi waktu itu sangat penting untuk menghindari terburu-buru saat mendekati *deadline*. Selain itu, melalui tugas besar ini dapat lebih memahami mengenai *breadth-first search* dan *depth-first search*, serta lebih mengenal bahasa pemrograman C#.

DAFTAR PUSTAKA

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

LAMPIRAN

Link Repository : https://github.com/syauqijan/Tubes2_krustycrew
Link Video Demo : <https://bit.ly/VideoKrustycrew>