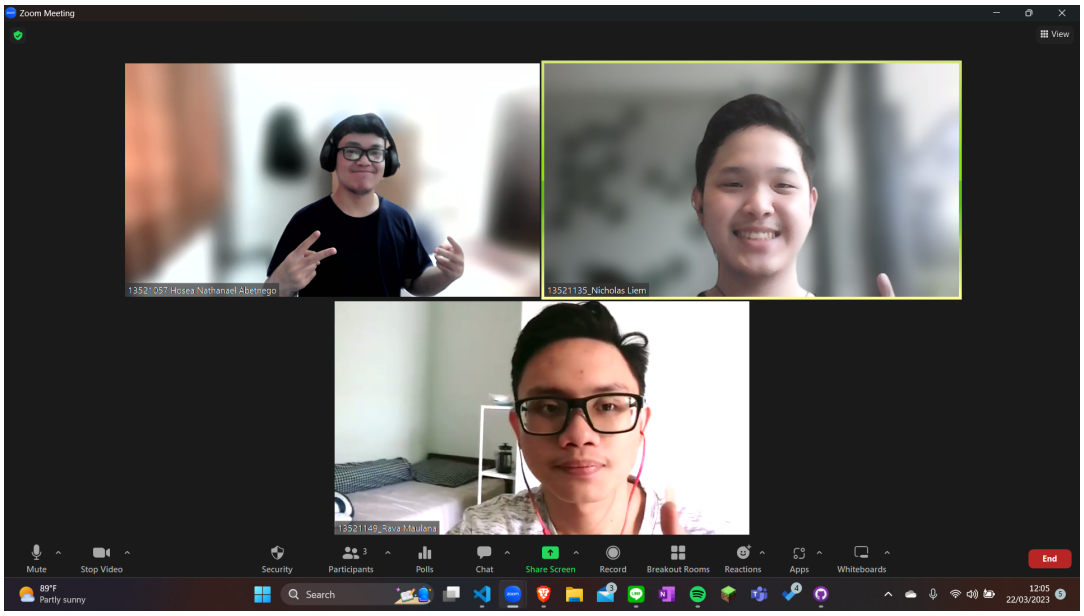
IF2211 - Strategi Algoritma

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

Laporan Tugas Besar II



Nama	NIM
Hosea Nathanael Abetnego	13521057
Nicholas Liem	13521135
Rava Maulana	13521149

Institut Teknologi Bandung
Sekolah Teknik Elektro dan Informatika

Tahun Ajaran 2022/2023

Daftar Isi

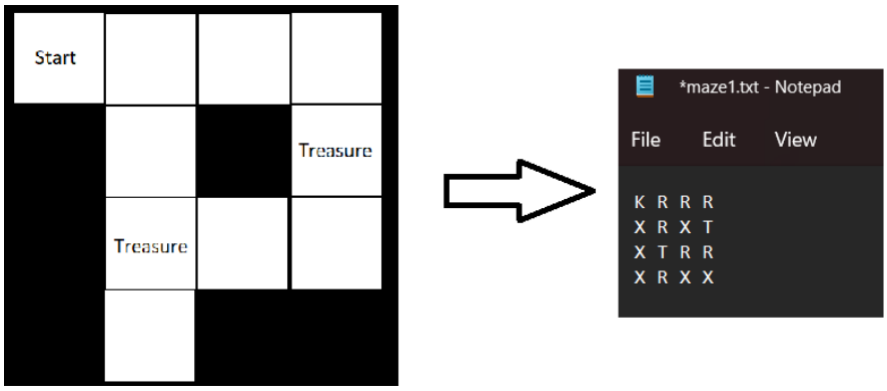
1 Deskripsi Masalah

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

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

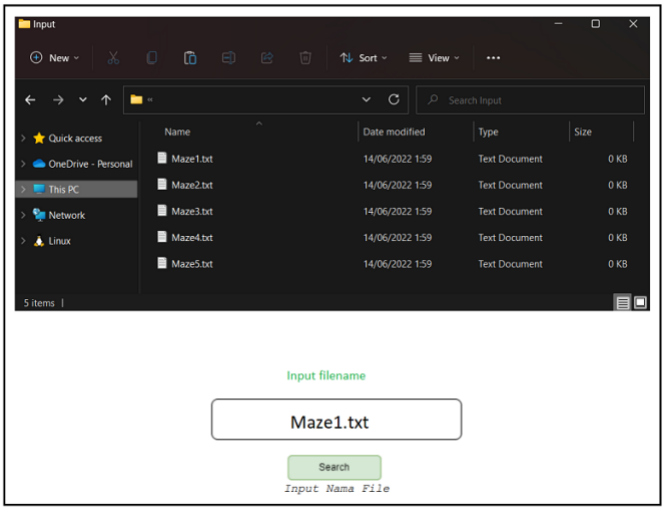
- K : Krusty Krab (Titik Awal)
- T : Treasure
- R : Grid yang mungkin diakses / sebuah lintasan
- X : Grid halangan yang tidak dapat diakses

Contoh file input:



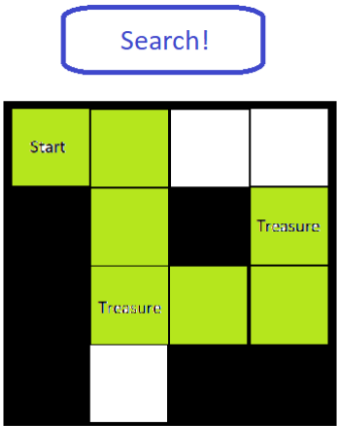
Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), anda dapat menelusuri grid (simpul) yang mungkin dikunjungi hingga ditemukan rute solusi, baik secara melebar ataupun mendalam bergantung alternatif algoritma yang dipilih. Rute solusi adalah rute yang memperoleh seluruh treasure pada maze. Perhatikan bahwa rute yang diperoleh dengan algoritma BFS dan DFS dapat berbeda, dan banyak langkah yang dibutuhkan pun menjadi berbeda. Prioritas arah simpul yang dibangkitkan dibebaskan asalkan ditulis di laporan ataupun readme, semisal LRUD (left right up down). Tidak ada pergerakan secara diagonal. Anda juga diminta untuk memvisualisasikan input txt tersebut menjadi suatu grid maze serta hasil pencarian rute solusinya. Cara visualisasi grid dibebaskan, sebagai contoh dalam bentuk matriks yang ditampilkan dalam GUI dengan keterangan berupa teks atau warna. Pemilihan warna dan maknanya dibebaskan ke masing - masing kelompok, asalkan dijelaskan di readme / laporan

Contoh input aplikasi:



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.

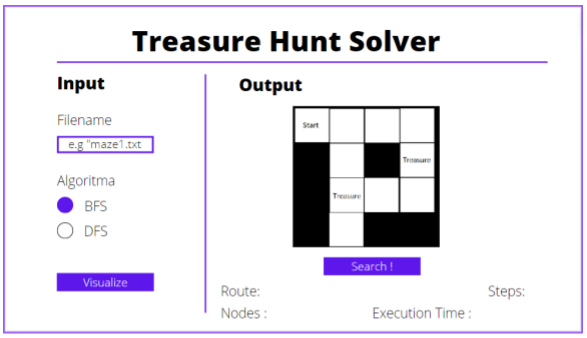
Contoh output aplikasi:



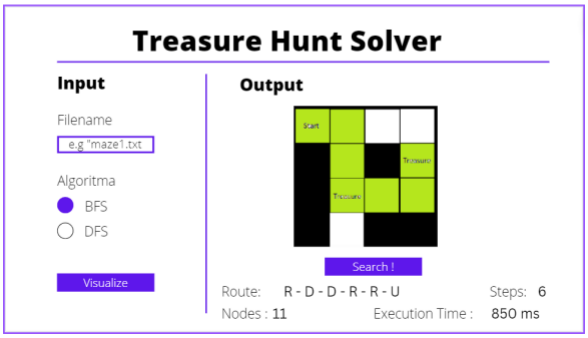
Gambar 4. Contoh output program untuk gambar 2

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.

1.2 Spesifikasi Program



Gambar 5. Tampilan Program Sebelum dicari solusinya

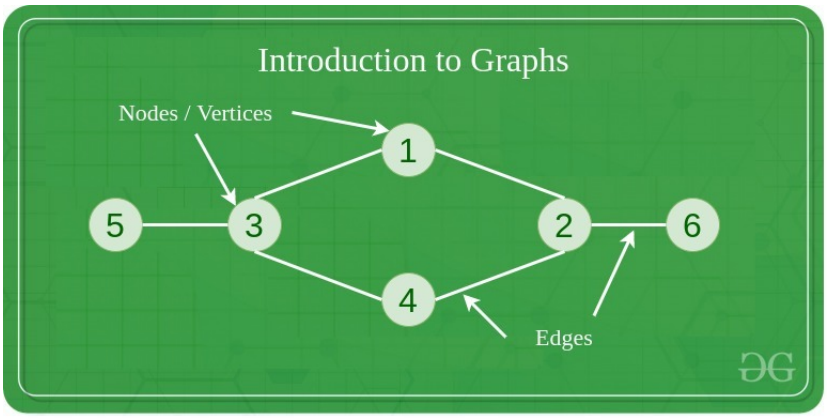


1. Masukan program adalah file maze treasure hunt tersebut atau nama filenya.
2. Program dapat menampilkan visualisasi dari input file maze dalam bentuk grid dan pewarnaan sesuai deskripsi tugas.
3. Program memiliki toggle untuk menggunakan alternatif algoritma BFS ataupun DFS.
4. Program memiliki tombol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

2 Landasan Teori

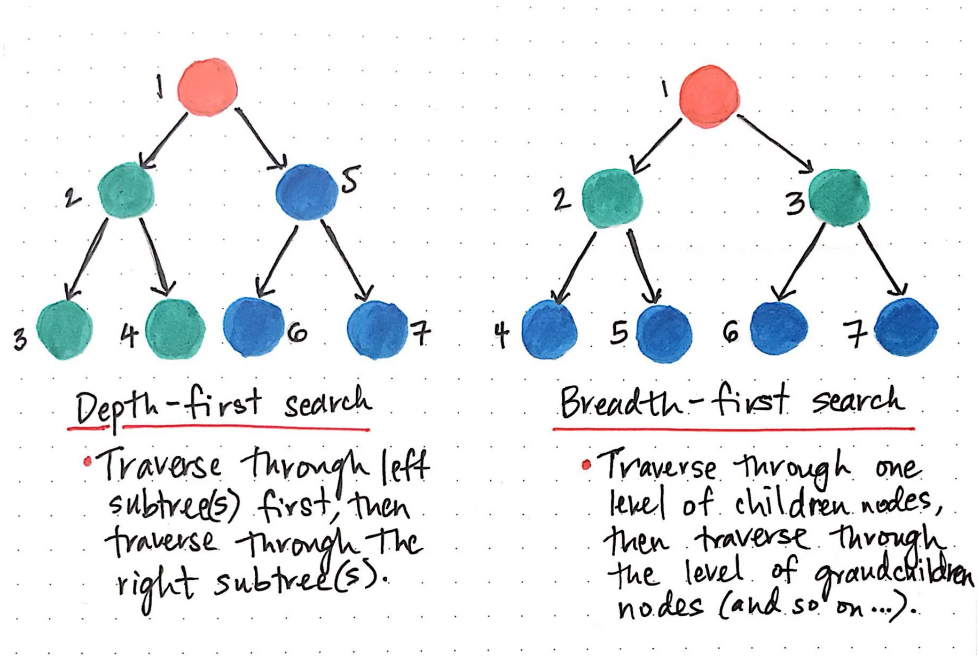
2.1 Traversal Graf, DFS, BFS

2.1.1 Traversal Graf



Sumber: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Sebuah graf terdiri dari dua komponen utama, yakni simpul dan sisi. Simpul biasanya disebut sebagai *node* dan sisi sebagai *edges*. Lalu, sebuah graf yang lengkap adalah semua simpul yang terhubung antara satu sama lain melalui suatu sisi. Pada penggunaannya, sebuah graf merupakan suatu himpunan simpul dan himpunan sisi yang dapat direpresentasikan dengan berbagai cara salah satunya adalah *adjacency matrix*, *adjacency list*, dan *adjacency set*. Pada program ini, tipe representasi graf yang digunakan adalah *adjacency list*.

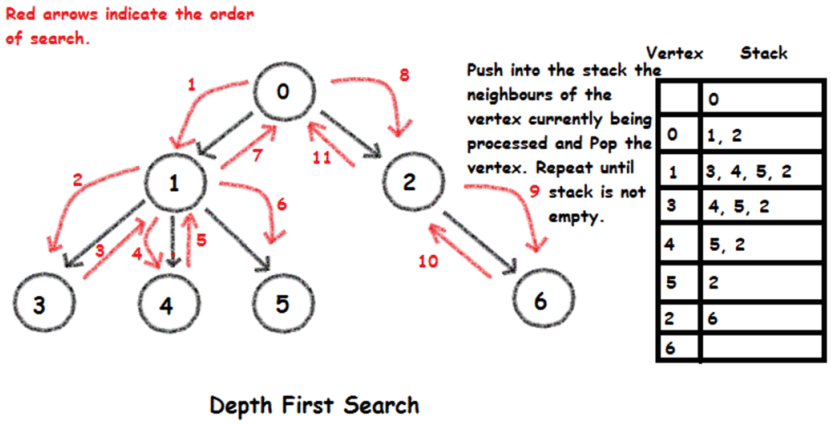


Sumber:
https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9

Bagaimana traversal graf bekerja pada representasi *adjacency list* adalah setiap simpul disimpan pada suatu *hash set* yang terdiri dari simpul sebagai *key* dan sebuah

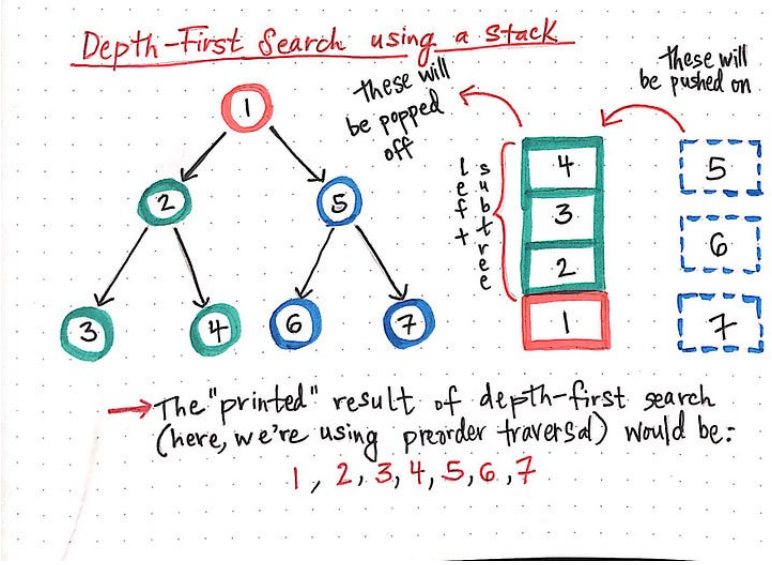
value berupa *list* berisi simpul-simpul lain yang bersisian dengan simpul *key*. Dengan representasi ini, traversal graf dapat didengan mudah dilakukan dengan mengiterasi simpul-simpul setiap dari *list* pada *hash set* tersebut. Untuk mekanisme teknis dari traversal graf, terdapat dua umum yang dipakai, yakni *Depth First Search* dan *Breadth First Search*. Masing-masing teknik memiliki keunikannya masing-masing yang akan dibahas di bawah ini.

2.1.2 Depth First Search (DFS)



Sumber:
http://www.crazyforcode.com/depth-first-search-dfs-algorithms-data-structures/

Salah satu teknik traversal graf yang umumnya dipakai adalah *Depth First Search* atau biasanya juga disebut sebagai pencarian mendalam. Teknik ini biasanya dapat diimplementasikan dengan dua cara, yakni pencarian secara rekursif atau menggunakan struktur data *stack*. Masing-masing memiliki keunikannya sendiri tetapi satu karakteristik yang membedakan teknik pencarian ini dengan yang lain adalah bagaimana teknik ini dapat melakukan proses *backtracking*.

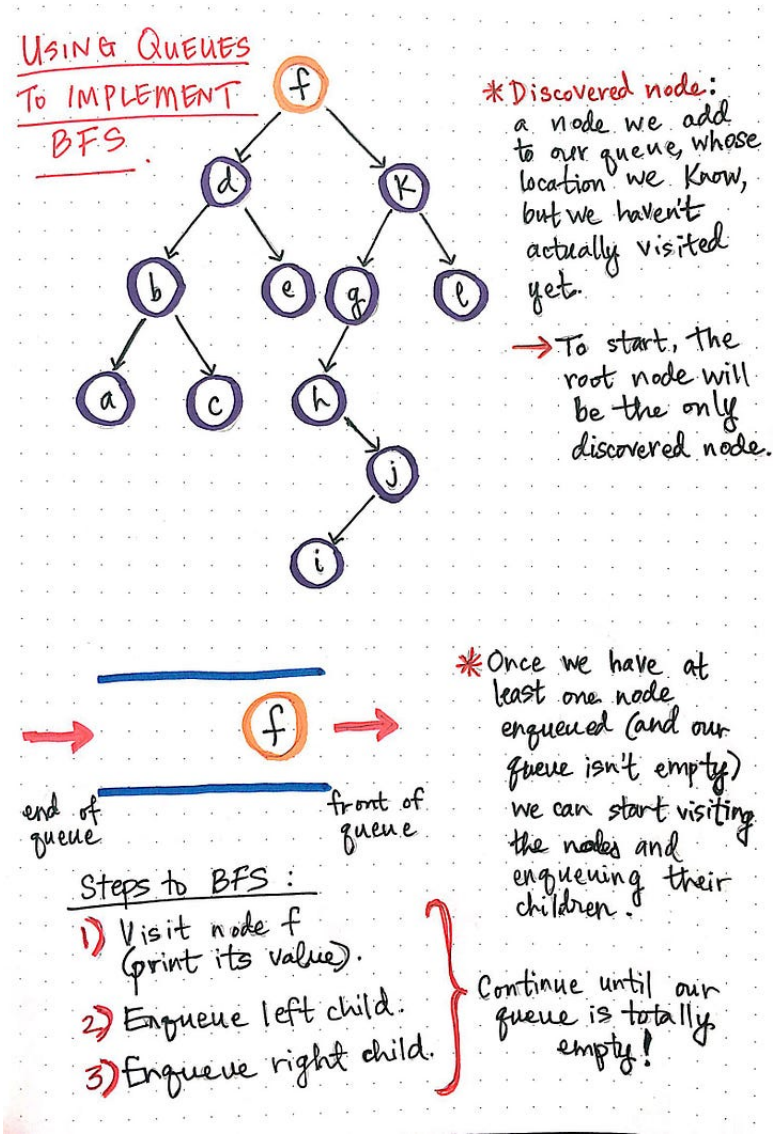


Sumber:
https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9

DFS bekerja dengan memulai traversal pada simpul *root*, kemudian berdasarkan prioritasnya (bisa dalam bentuk angka, alfabetikal, arah mata angin, arah gerak, dan sebagainya) memasukkan simpul lain yang bersisian pada *stack* kemudian mengulang iterasi tersebut hingga menemukan simpul yang dicari. Tetapi, jika suatu cabang simpul mencapai suatu *dead end* maka DFS akan melakukan *backtracking* ke simpul yang tersisa di *stack* yakni simpul-simpul yang belum dikunjungi sebelumnya.

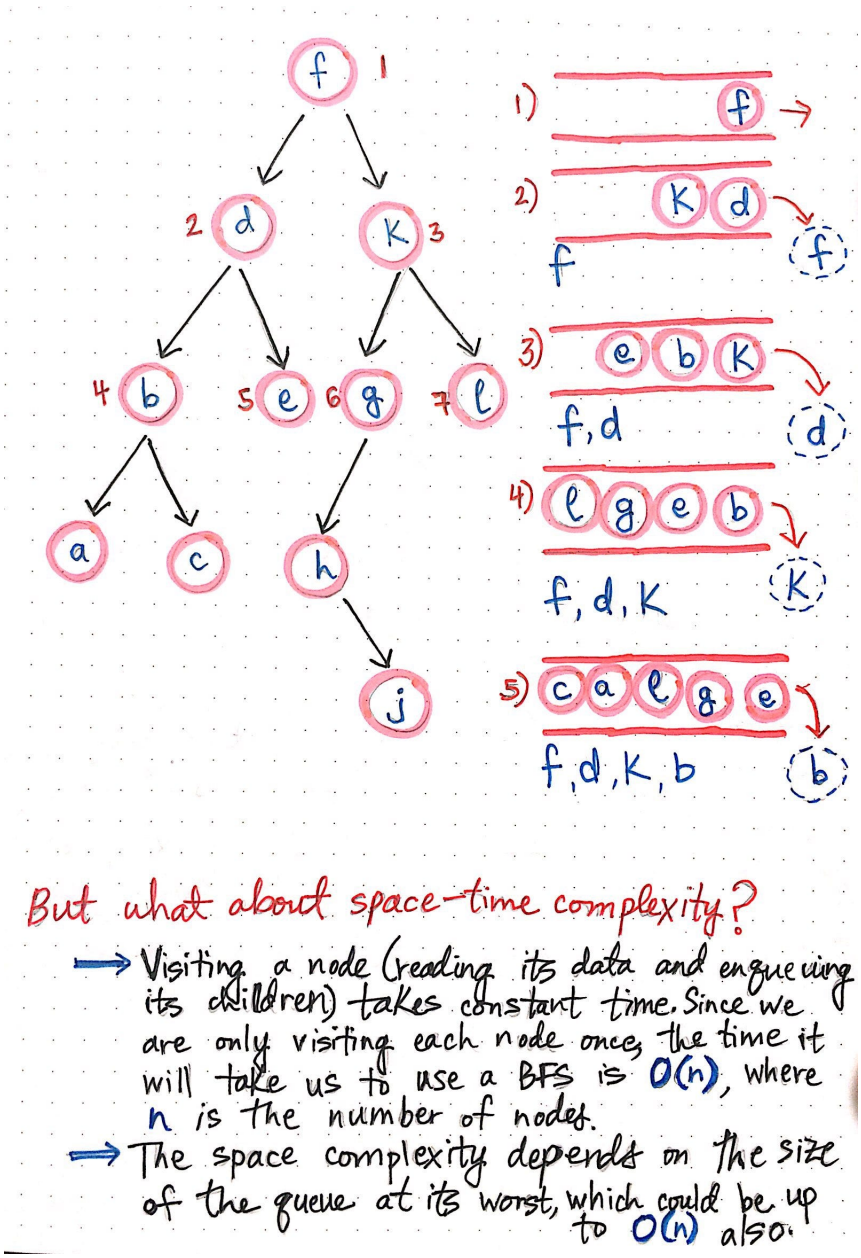
2.1.3 Breadth First Search (BFS)

Teknik traversal graf lain selain DFS adalah *Breadth First Search* atau biasanya disebut sebagai pencarian menyeluruh. Teknik ini diimplementasikan menggunakan struktur data *queue*.



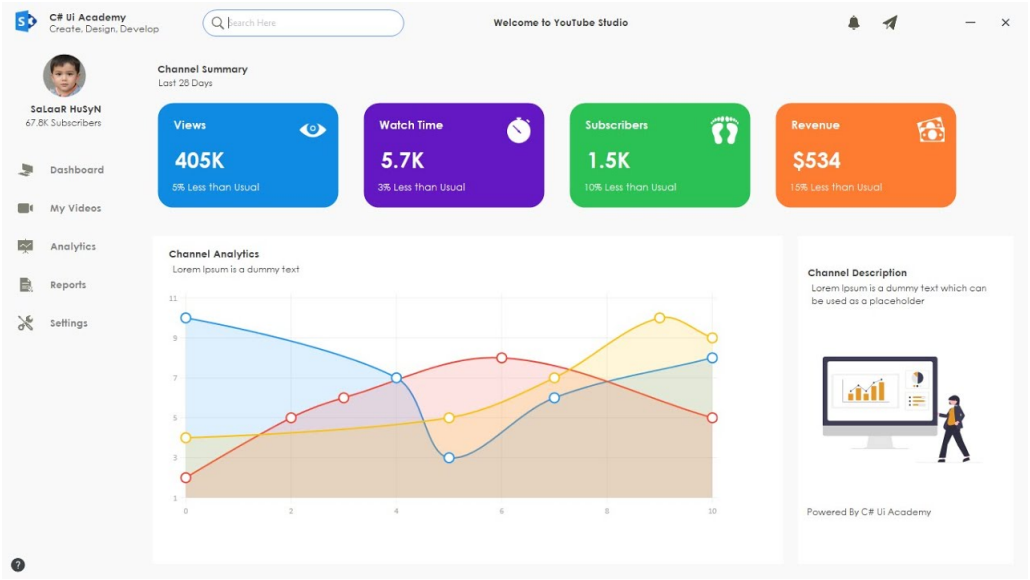
Sumber:
https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9

Berbeda dengan DFS, teknik traversal BFS menggunakan struktur data *queue* di mana semua simpul tetangga akan didaftarkan apada *queue* sehingga proses ini akan melakukan pencarian secara menyeluruh dan satu persatu. Setiap simpul yang dikunjungi dan memiliki tetangga akan didaftarkan setiap tetangganya ke dalam *queue* dan proses pencarian selalu dimulai dari awal *queue*. Oleh karena menggunakan struktur data *queue* teknik pencarian ini tidak memiliki karakteristik *backtracking*.



Sumber:
https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9

2.2 C# Desktop Application Development



Sumber: https://www.youtube.com/watch?v=qW8kUTadRpw

C# adalah bahasa pemrograman yang dikembangkan oleh Microsoft dan digunakan secara luas dalam pengembangan aplikasi berbasis Windows. Dalam pengembangan aplikasi C#, pengembang dapat menggunakan banyak alat bantu seperti Visual Studio, .NET Framework, dan Xamarin untuk membuat aplikasi desktop, web, dan mobile. Dalam pengembangan aplikasi desktop, C# biasanya digunakan untuk membuat aplikasi Windows Forms, WPF (Windows Presentation Foundation), dan UWP (Universal Windows Platform). Sementara itu, untuk pengembangan aplikasi web, C# digunakan untuk membuat aplikasi ASP.NET dan ASP.NET Core. Selain itu, C# juga digunakan dalam pengembangan game dengan menggunakan platform seperti Unity. Penggunaan C# dalam pengembangan aplikasi sangat populer karena bahasa pemrograman ini memiliki sintaks yang mudah dipahami, memiliki performa yang baik, dan terintegrasi dengan baik dengan banyak alat bantu pengembangan.

3 Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

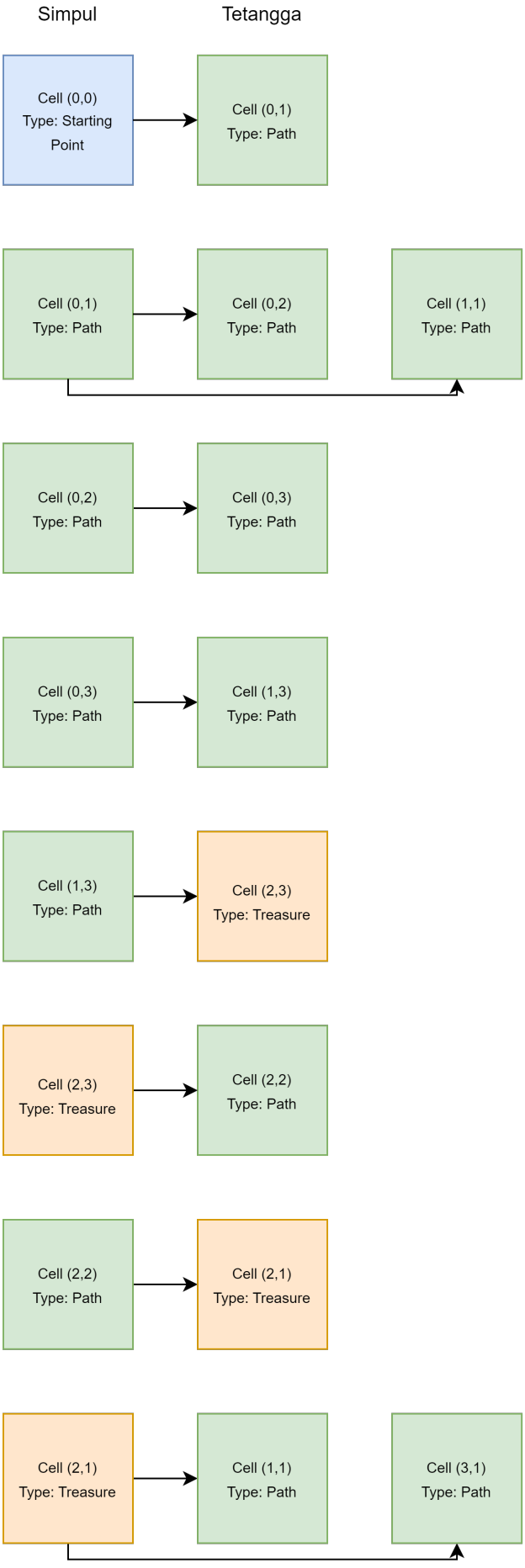
Langkah-langkah pemecahan yang kami pakai adalah sebagai berikut:

1. Pembentukan *map* dengan pembacaan file.
2. Pembentukan graf berdasarkan *map*.
3. Melakukan penelurusan berdasarkan tipe pencarian.

3.2 *Mapping* Persoalan

Cell (0,0) Type: Starting Point	Cell (0,1) Type: Path	Cell (0,2) Type: Path	Cell (0,3) Type: Path
Cell (1,0) Type: Wall	Cell (1,1) Type: Path	Cell (1,2) Type: Wall	Cell (1,3) Type: Path
Cell (2,0) Type: Wall	Cell (2,1) Type: Treasure	Cell (2,2) Type: Path	Cell (2,3) Type: Treasure
Cell (3,0) Type: Wall	Cell (3,1) Type: Path	Cell (3,2) Type: Wall	Cell (3,3) Type: Wall

Suatu peta akan direpresentasikan sebagai suatu matriks yang setiap isinya merupakan sebuah *cell*. Setiap *cell* memiliki tipenya masing-masing, entah itu tembok, jalan (*path*), ataupun *treasure*. Kemudian, akan disiapkan suatu graf yang dibentuk dari *cell-cell* yang terletak dalam matriks. Pembentukan graf akan dimulai dari titik $(0, 0)$ kemudian sampai titik (m, n) untuk peta berukuran $m \times n$. Pemetaannya adalah setiap *cell* kanan, kiri, atas, dan bawah akan dicek tipenya jika tipenya adalah tembok, maka *cell* tersebut tidak akan dimasukkan ke dalam *adjacency list* dari *cell* tersebut. Kemudian, berdasarkan pilihan dari pengguna, akan dilakukan proses traversal graf berdasarkan tipe-tipe yang ada.


Simpul
Tetangga
Cell (0,0)
Type: Starting
Point
Cell (0,1)
Type: Path
Cell (0,1)
Type: Path
Cell (0,2)
Type: Path
Cell (1,1)
Type: Path
Cell (0,2)
Type: Path
Cell (0,3)
Type: Path
Cell (0,3)
Type: Path
Cell (1,3)
Type: Path
Cell (1,3)
Type: Path
Cell (2,3)
Type: Treasure
Cell (2,3)
Type: Treasure
Cell (2,2)
Type: Path
Cell (2,2)
Type: Path
Cell (2,1)
Type: Treasure
Cell (2,1)
Type: Treasure
Cell (1,1)
Type: Path
Cell (3,1)
Type: Path

3.3 Ilustrasi Kasus Lain

Berikut adalah visualisasi atau ilustrasi dari *test case* 4.

Cell (0,0) Type: Starting Point	Cell (0,1) Type: Path	Cell (0,2) Type: Path	Cell (0,3) Type: Path	Cell (0,4) Type: Path	Cell (0,5) Type: Path	Cell (0,6) Type: Path	Cell (0,7) Type: Wall
Cell (1,0) Type: Wall	Cell (1,1) Type: Wall	Cell (1,2) Type: Path	Cell (1,3) Type: Wall	Cell (1,4) Type: Path	Cell (1,5) Type: Wall	Cell (1,6) Type: Path	Cell (1,7) Type: Wall
Cell (2,0) Type: Wall	Cell (2,1) Type: Treasure	Cell (2,2) Type: Path	Cell (2,3) Type: Wall	Cell (2,4) Type: Wall	Cell (2,5) Type: Wall	Cell (2,6) Type: Path	Cell (2,7) Type: Wall
Cell (3,0) Type: Wall	Cell (3,1) Type: Wall	Cell (3,2) Type: Path	Cell (3,3) Type: Path	Cell (3,4) Type: Path	Cell (3,5) Type: Path	Cell (3,6) Type: Path	Cell (3,7) Type: Wall
Cell (4,0) Type: Wall	Cell (4,1) Type: Wall	Cell (4,2) Type: Wall	Cell (4,3) Type: Wall	Cell (4,4) Type: Path	Cell (4,5) Type: Wall	Cell (4,6) Type: Path	Cell (4,7) Type: Wall
Cell (5,0) Type: Wall	Cell (5,1) Type: Wall	Cell (5,2) Type: Wall	Cell (5,3) Type: Path	Cell (5,4) Type: Path	Cell (5,5) Type: Wall	Cell (5,6) Type: Treasure	Cell (5,7) Type: Wall
Cell (6,0) Type: Wall	Cell (6,1) Type: Wall	Cell (6,2) Type: Wall	Cell (6,3) Type: Wall	Cell (6,4) Type: Wall	Cell (6,5) Type: Wall	Cell (6,6) Type: Wall	Cell (6,7) Type: Wall

4 Analisis Pemecahan Masalah

4.1 Pseudocode Program Utama

4.1.1 Depth First Search

Deskripsi Program: Depth First Search
Program akan mengembalikan suatu *list* yang berupa urutan *cell* dari titik awal permainan hingga titik akhir ketika semua harta karun telah ditemukan. **Urutan pencarian DFS: L - D - R - U** atau mengikuti urutan dari pendaftaran graf yang dapat dilihat pada Utils.cs bagian registerVertex.

```
1     function DepthFirstSearch(Graph graph) -> List<Cell>
2         Stack<Cell> paths <- new Stack<Cell>()
3         Stack<Cell> availableNodes <- new Stack<Cell>()
4         HashSet<Cell> visitedNodes <- new HashSet<Cell>()
5         Stack<Cell> candidatePath <- new Stack<Cell>()
6
7         availableNodes.Push(graph.EntryVertex)
8
9         while (availableNodes.Count > 0) do
10            Cell currentCell <- availableNodes.Pop()
11            visitedNodes.Add(currentCell)
12            candidatePath.Push(currentCell)
13            paths.Push(currentCell)
14            currentCell.addVisitedCount()
15
16            If (candidatePathHasAllTreasures(candidatePath)) then
17                -> candidatePath
18            End If
19
20            bool isDeadEnd <- true
21            List<Cell> edges <- graph.GetCellNeighbors(currentCell)
22            For Each (Cell cell in edges) do
23                If (not visitedNodes.Contains(cell) and not
    availableNodes.Contains(cell)) then
24                    availableNodes.Push(cell)
25                    isDeadEnd <- false
26            End For
27
28            while(isDeadEnd and paths.Count > 0) do
29                Cell lastCell <- paths.Pop()
30                candidatePath.Push(lastCell)
31                lastCell.addVisitedCount()
32                List<Cell> lastCellEdges <- graph.GetCellNeighbors(
    lastCell)
33                For Each (Cell cell in lastCellEdges) do
34                    If (not paths.Contains(cell) and not
    visitedNodes.Contains(cell))
35                        isDeadEnd <- false
36                    End If
37                End For
38
39        -> new List<Cell>()
```

4.1.2 Breadth First Search

Deskripsi Program: Breadth First Search
Program akan mengembalikan suatu *list* yang berupa urutan *cell* dari titik awal permainan hingga titik akhir ketika semua harta karun telah ditemukan.

```
1    function BreadthFirstSearch(ByVal graph As Graph, ByVal start
     As Cell, ByVal treasureCount As Integer, ByVal tsp As Boolean,
     ByVal Optional type As Integer = 9) As List(Of Cell)
2
3        Dim checkedCells As List(Of Cell) = New List(Of Cell)()
4        Dim checkQueue As Queue(Of List(Of Cell)) = New Queue(Of
     List(Of Cell))()
5        Dim checkPath As List(Of Cell) = New List(Of Cell)()
6        checkPath.Add(start)
7        checkQueue.Enqueue(checkPath)
8        While checkQueue.Count \> 0
9            Dim currCellList As List(Of Cell) = checkQueue.Dequeue
     ()
10           Dim currCell As Cell = currCellList.ElementAt(
     currCellList.Count - 1)
11
12           If checkedCells.Contains(currCell) Then
13               Continue While
14           End If
15
16           checklist.Add(currCell)
17           checkedCells.Add(currCell)
18           currCell.VisitedCount += 1
19
20           If currCell.Type = type AndAlso Not currCell.isEqual(
     start) AndAlso Not solutionSpace.Contains(currCell) Then
21               treasureFound += 1
22               solutionSpace.Add(currCell)
23
24               If treasureFound < treasureCount Then
25                   Dim nextPath As List(Of Cell) =
     BreadthFirstSearch(graph, currCell, treasureCount, tsp)
26
27                   For i As Integer = 1 To nextPath.Count - 1
28                       currCellList.Add(nextPath(i))
29                   Next
30               ElseIf tsp Then
31                   Dim findHome As List(Of Cell) =
     BreadthFirstSearch(graph, currCell, treasureCount, False, 0)
32
33                   For i As Integer = 1 To findHome.Count - 1
34                       currCellList.Add(findHome(i))
35                   Next
36               End If
```

```
37
38                  Return currCellList
39              End If
40
41              For Each neighbor In graph.GetCellNeighbors(currCell)
42
43                  If Not checkedCells.Contains(neighbor) Then
44                      Dim newPath As List(Of Cell) = New List(Of Cell
    )()
45
46                      For Each elmt In currCellList
47                          newPath.Add(elmt)
48                      Next
49
50                      newPath.Add(neighbor)
51                      checkQueue.Enqueue(newPath)
52                  End If
53              Next
54          End While
55
56          Return checkPath
```

4.2 Struktur Data dan Kelas

4.2.1 Cell

Kelas Cell
Tujuan Pembuatan: Pembuatan kelas *Cell* ditujukan untuk mengatur setiap blok sel yang akan disimpan pada suatu larik atau dalam kelas lain supaya gampang untuk diindentifikasi apakah suatu petak adalah *path*, harta karun, ataupun tembok.
Daftar-daftar atribut:

1. Integer: Row
2. Integer: Col
3. Integer: Type
4. Integer: VisitedCount

Penjelasan Tambahan:
Nilai integer untuk tembok adalah 3, nilai integer untuk harta karun adalah 9, nilai harta karun untuk jalan adalah 1.

4.2.2 Map

Kelas Map
Tujuan Pembuatan:
Pembuatan kelas *Map* ditujukan untuk mengatur setiap *Cell* yang dikonversi dari txt file menjadi kelas yang baru. Selain itu, kelas *Map* juga bertujuan untuk menyimpan

lokasi harta karun serta jumlah harta karun pada suatu peta.
Daftar-daftar atribut:

1. Array of Cells: Cells
2. Integer (Static): TreasureCount
3. Graph: graph
4. Integer: RowSize
5. Integer: ColZise
6. HashSet of Cells: TreasureCells

4.2.3 Graph

Kelas Graph
Tujuan Pembuatan:
Memudahkan untuk menyimpan suatu *node* dengan *node* lain menggunakan metode representasi *adjacency list*.
Daftar-daftar atribut:

1. Dictionary of Cell and List¡Cell¿: adjacencyList
2. Cell: entryVertex

4.2.4 Stack

Struktur Data Stack
Tujuan Pembuatan:
Penggunaan struktur data *stack* ditujukan untuk membantu operasional dari algoritma *Depth First Search* karena karakteristik utama dari struktur data ini membantu proses *backtracking* dari algoritma DFS.
Penggunaan:

1. paths (Algorithms/DepthFirstSearch.cs)
2. availableNodes (Algorithms/DepthFirstSearch.cs)
3. candidatePath (Algorithms/DepthFirstSearch.cs)

4.2.5 Queue

Struktur Data Queue
Tujuan Pembuatan:
Penggunaan struktur data *queue* ditujukan untuk membantu operasional dari algoritma *Breadth First Search*.
Penggunaan:

1. checkQueue (Algorithms/BreadthFirstSearch.cs)

4.2.6 HashSet

Struktur Data HashSet
Tujuan Pembuatan:
Sebagai pengganti *list* untuk memudahkan pencarian dari suatu *item*.
Penggunaan:

1. visitedNodes (Algorithms/DepthFirstSearch.cs)
2. hashSet (Graph/Graph.cs)
3. treasureCells (Map/Map.cs)

4.2.7 List

Struktur Data List
Tujuan Pembuatan:
Sebagai penampung *item* yang cukup efisien, digunakan diberbagai tempat.

4.2.8 Class Algorithms

1. Utils: Perkakas untuk membantu keberjalanan program, algoritma-algoritma penting yang ditaruh di dalam kelas ini adalah findEntryPoint, findTreasurePositions, dan registerVertex.
2. FileReader: Proses pembacaan file menjadi kelas-kelas dalam program yang dapat digunakan.
3. GraphException: *Exception handling* untuk kesalahan pembuatan graf.
4. FileReaderException: *Exception handling* untuk kesalahan pembacaan file.

4.3 Penggunaan Program

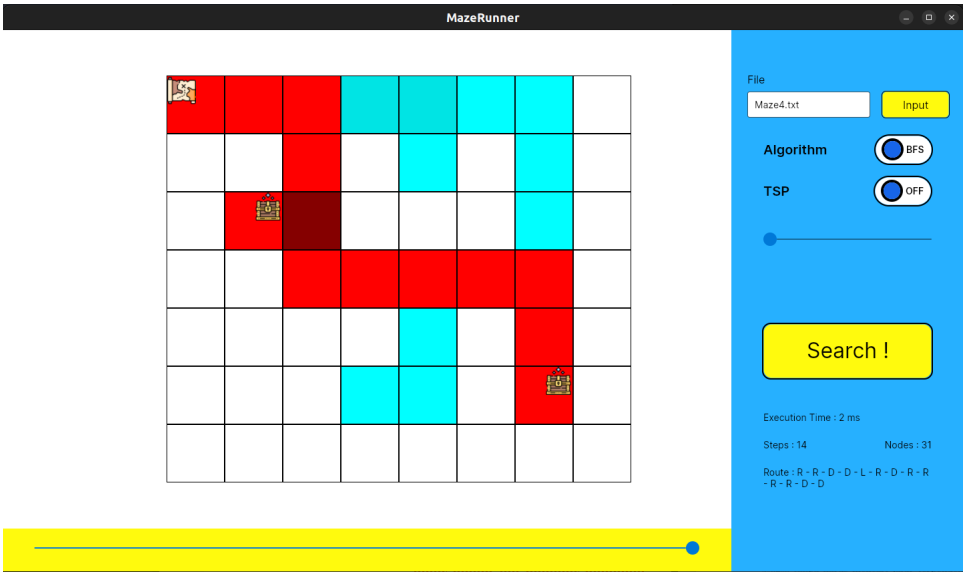
1. Masukkan nama file yang valid (file sudah berada pada folder `config`)
2. Jika nama file valid, tekan tombol `input` dan program akan menampilkan *maze* pada layar
3. Pilih algoritma yang digunakan (BFS/DFS) menggunakan *toggle* `algorithm`
4. Jika algoritma yang dipilih adalah *Breadth First Search*, pilih apakah program akan mencari rute TSP atau tidak menggunakan *toggle* `TSP`
5. Atur kecepatan animasi pencarian dengan menggunakan *slider* pada bagian kanan layar
6. Setelah pengaturan awal sudah dilakukan, tekan tombol `search` untuk memulai pencarian rute
7. Data hasil pencarian program akan ditampilkan pada bagian bawah layar

8. Langkah-langkah pencarian dapat disimulasikan secara manual dengan menggunakan *slider* pada bagian bawah layar

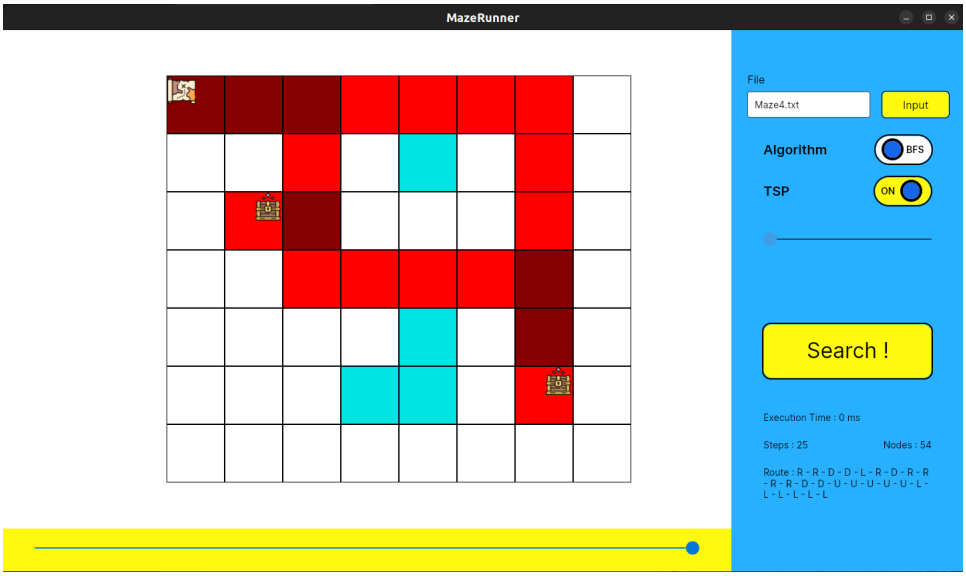
9. Ulangi langkah pertama jika ingin mencari rute untuk konfigurasi *maze* lainnya

4.4 Hasil Pengujian

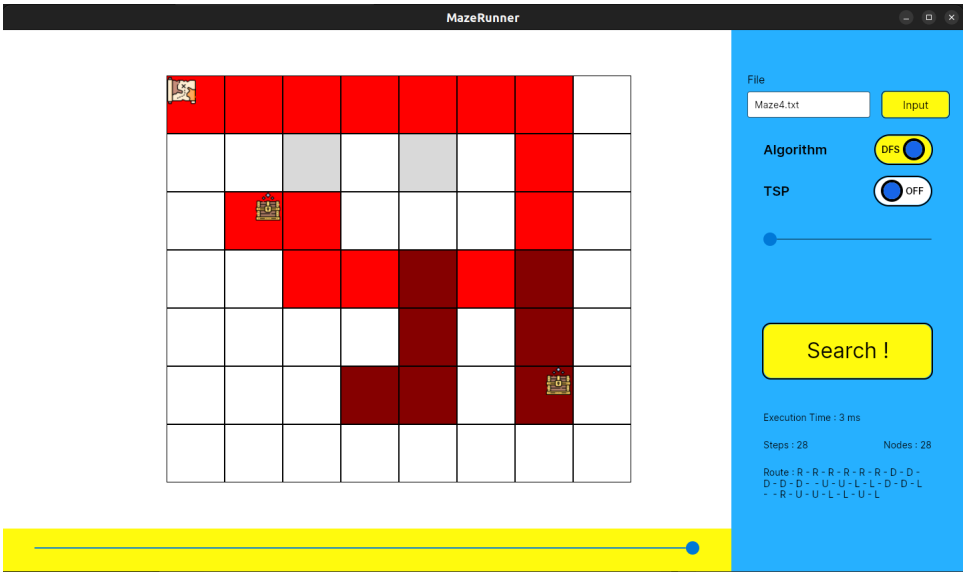
Pengujian dilakukan pada file `Maze4.txt` untuk menguji algoritma *Breadth First Search*, *Depth First Search*, dan *Travelling Salesman Problem*. Lokasi dari setiap *treasure* dilambangkan dengan gambar peti harta karun dan lokasi *start* dilambangkan dengan gambar peta. Pada layar, kotak berwarna putih merupakan kotak yang tidak dapat dilalui, kotak berwarna biru merupakan kotak yang telah dicek oleh algoritma, kotak berwarna abu-abu merupakan kotak yang tidak dicek oleh algoritma, dan kotak berwarna merah merupakan rute yang telah ditemukan. Semakin gelap warna kotak, semakin sering kotak tersebut dilewati oleh rute pencarian.



Hasil pencarian rute menggunakan algoritma *Breadth First Search* menghasilkan rute R - R - D - D - L - R - D - R - R - R - R - D - D. Waktu eksekusi algoritma adalah 2 milisekon. Algoritma memeriksa sebanyak 31 sel dan menghasilkan rute solusi sebanyak 14 sel.



Hasil pencarian rute menggunakan algoritma *Breadth First Search* untuk permasalahan *Travelling Salesman Problem* menghasilkan rute R - R - D - D - L - R - D - R - R - R - R - D - D - U - U - U - U - U - L - L - L - L - L. Waktu eksekusi algoritma adalah dibawah 1 milisekon. Algoritma memeriksa sebanyak 54 sel dan menghasilkan rute solusi sebanyak 25 sel.



Hasil pencarian rute menggunakan algoritma *Depth First Search* menghasilkan rute R - R - R - R - R - R - D - D - D - D - D - U - U - L - L - D - D - L - R - U - U - L - L - U - L. Waktu eksekusi algoritma adalah 3 milisekon. Algoritma memeriksa sebanyak 28 sel dan menghasilkan rute solusi sebanyak 28 sel.

4.5 Analisis Desain Solusi

Dalam masalah pencarian jalur di dalam sebuah *maze* yang memiliki beberapa harta karun, kami menggunakan dua pendekatan traversal graf yakni *Depth First Search*

dan *Breadth First Search*. Kedua metode ini memiliki kelebihan dan kekurangan masing-masing tergantung pada kondisi *map* atau objektif dari penggunaannya.

Jika objektif program adalah untuk menemukan **jarak terpendek** dari posisi awal ke semua harta karun, algoritma BGS akan membangun jalur secara teratur dan berurutan dari posisi awal dan menelusuri semua cabang yang mungkin ada secara seimbang.

Jika objektif program adalah untuk menemukan jalur ke semua harta karun dengan **tanpa harus mencari jalur terpendek**, algoritma DFS adalah pilihan yang tepat. Selain itu, penggunaan DFS pada kasus umum *multiple* harta karun mungkin tidak seefektif jika harta karunnya hanya satu sebab kita tidak tahu lokasi di mana harta karun disembunyikan. Oleh sebab itu, pencarian menggunakan DFS mungkin akan menghasilkan jalur yang lebih panjang dan memerlukan waktu yang lebih lama untuk menemukan semua harta karun.

Dalam kesimpulannya, penggunaan **BFS lebih disarankan** dalam masalah maze dengan ***multiple*** **harta karun** karena algoritma ini akan mencari **jalur terpendek ke semua harta karun dalam maze secara efisien**. Sementara itu, **DFS** dapat digunakan jika tujuan utama adalah menemukan semua jalur yang mungkin ke setiap harta karun dalam maze, tetapi algoritma ini mungkin **tidak efektif dalam hal waktu dan memori**.

5 Kesimpulan dan Saran

5.1 Kesimpulan

Dari tugas besar ini, dapat disimpulkan bahwa kami berhasil mengembangkan sebuah aplikasi dengan GUI sederhana yang dapat mengimplementasikan algoritma BFS dan DFS untuk mendapatkan rute memperoleh seluruh treasure atau harta karun yang ada pada maze. Kami dapat membaca input file txt dengan spesifikasi maze yang telah ditentukan. Kami juga belajar bagaimana cara memvisualisasikan grid maze serta hasil pencarian rute solusinya dalam bentuk matriks yang ditampilkan dalam GUI dengan keterangan berupa teks atau warna.

5.2 Saran

Selain itu, kami juga memberikan saran untuk pengembangan selanjutnya. Pertama, kami menyarankan agar kode program yang digunakan dalam tugas ini dapat dikembangkan lebih lagi, baik dari sisi strategi algoritma, modularitas, dan lain sebagainya. Kedua, strategi algoritma BFS dan DFS yang kami implementasikan mungkin belum menjadi strategi yang terbaik, oleh karena itu perlu dilakukan eksplorasi dan pengembangan lebih lanjut.

5.3 Refleksi

Kami juga memberikan komentar mengenai pengalaman kami dalam mengerjakan tugas besar ini. Kami merasa senang karena dapat menyelesaikan tugas ini dan belajar banyak hal baru terkait algoritma BFS dan DFS. Kami juga merasa sedikit kesulitan dalam memvisualisasikan grid maze dan rute solusi pada GUI, tetapi kami berhasil menyelesaikannya dengan baik.

5.4 Tanggapan

Dalam refleksi kami, kami belajar bagaimana cara mengimplementasikan algoritma BFS dan DFS dalam bahasa pemrograman C#. Kami juga belajar untuk mengelola waktu serta tanggung jawab yang kami miliki. Kami belajar untuk saling melengkapi dan menjaga komunikasi yang baik dalam kelompok, sehingga dapat menyelesaikan tugas dengan baik. Kami juga belajar bagaimana cara mengatasi kesulitan dan menemukan solusi dari permasalahan yang muncul.

6 Daftar Pustaka

Geeks, Geeks For. *Graph Data Structure And Algorithms*. 2023. URL: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/ (visited on 03/21/2023).

— *Graph Data Structure And Algorithms*. 2023. URL: Depth%20First%20Search%20or%20DFS%20for%20a%20Graph (visited on 03/21/2023).

Munir, Rinaldi. *Breadth/Depth First Search (BFS/DFS) (Bagian 1)*. 2023. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf (visited on 03/21/2023).

— *Breadth/Depth First Search (BFS/DFS) (Bagian 2)*. 2023. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf (visited on 03/21/2023).

7 Lampiran

7.1 Link Repository GitHub

Repo GitHub [Click Me!] or https://github.com/NicholasLiem/Tubes2_MazeRunner

7.2 Link Video (YouTube)

YouTube [Click Me!] or https://youtu.be/Ny6lmZ9eprA