Perbandingan Aplikasi Algoritma BFS dan DFS dalam Mencari Jalan Keluar dari *Perfect Maze*

Vincent Ho - 13520093
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13520093@std.stei.itb.ac.id

***Abstract*—Maze atau yang biasa dikenal dengan labirin merupakan kumpulan jalur rumit yang harus diselesaikan untuk mencapai titik goal dari titik awal. Seiring dengan perkembangan jaman, *maze* menjadi salah satu tipe *puzzle* yang sangat popular di dunia dan merupakan salah satu persoalan umum yang sering ditemukan dalam dunia informatika. Penyelesaian *Maze* ini dapat dilakukan dengan menganggapnya sebagai persoalan dalam teori graf atau *tree* dan diselesaikan dengan menggunakan algoritma pencarian seperti *Breadth-First Search*, *Depth-First Search*, *Iterative Deepening Search*, dan lain sebagainya. Pada makalah ini penulis akan membahas mengenai pencarian rute keluar dari sebuah persoalan *perfect maze* yang dimodelkan sebagai *tree* menggunakan algoritma BFS dan DFS.**

Keywords—BFS; DFS; Perfect Maze; labirin; tree; graph.

I. PENDAHULUAN

Perkembangan teknologi dan ilmu pengetahuan yang pesat menyebabkan banyak perubahan dalam dunia. Saat ini dunia telah memasuki era digital dimana hampir seluruh hal dapat diselesaikan dengan menggunakan teknologi. Persoalan yang sebelumnya sangat susah dan rumit kini dapat dengan mudah dan konsisten diselesaikan dengan memanfaatkan komputer atau kekuatan komputasi. Hal – hal yang sebelumnya memerlukan tenaga manusia pun secara perlahan dapat di otomatisasi dan digantikan oleh mesin, sehingga berjalan dengan lebih efektif dan efisien. Contohnya mesin ATM yang dapat melayani kebutuhan transaksi bank 24 jam tanpa perlu adanya teller yang bertugas.

Semakin berkembangnya dunia semakin banyak pula masalah yang muncul yang tidak dapat diselesaikan tanpa memanfaatkan teknologi atau tenaga komputasi. Oleh karena itu timbulah kebutuhan akan perangkat lunak atau program yang perlu dikembangkan untuk menyelesaikan masalah-masalah yang muncul tersebut. Setiap masalah memiliki pendekatan penyelesaian yang berbeda-beda menggunakan algoritma yang berbeda pula, disana lah tugas seorang *programmer* untuk menemukan algoritma atau cara yang paling tepat dalam menyelesaikan suatu persoalan dengan program.

Salah satu teori penting dalam dunia informatika adalah teori struktur *tree* atau pohon yang merupakan bentuk khusus dari graf. Banyak sekali persoalan dalam kehidupan nyata yang dapat dimodelkan ke dalam bentuk struktur pohon. Beberapa contohnya antara lain penyimpanan *file* dalam dalam sebuah *database*, klasifikasi makhluk hidup, struktur sebuah organisasi, pohon keturunan sebuah keluarga, *perfect maze*/labirin sempurna, dan lain-lain.

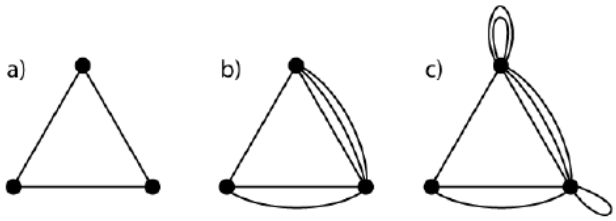
Pada makalah ini, penulis akan membahas secara spesifik mengenai aplikasi algoritma *Breadth-First Search* dan *Depth-First Search* dalam mencari jalan keluar dari *Perfect maze* atau labirin sempurna.

II. LANDASAN TEORI

A. Graf

Graf dalam struktur diskrit digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Sehingga secara sederhana graf didefinisikan sebagai kumpulan titik atau simpul (*node/vertex*) yang dihubungkan oleh garis-garis (*edge*).

Graf G didefinisikan dengan notasi G = (V, E) yang dalam hal ini V adalah himpunan tidak kosong dari simpul-simpul (*vertices*) dan E adalah himpunan sisi (*edges*) yang menghubungkan sepasang simpul. $V = \{v_1, v_2, v_3, \ldots, v_n\}$. $E = \{e_1, e_2, \ldots, e_n\}$.

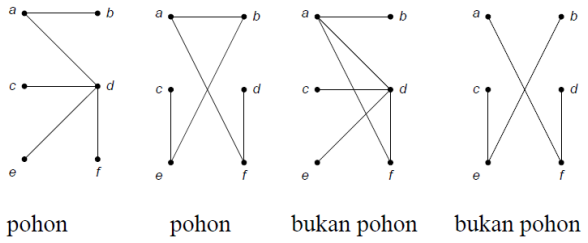


Gambar 1. (a) Graf sederhana. (b) Graf ganda. (c) Graf semu.
Sumber: Graf Bagian 1 oleh Rinaldi Munir

B. Pohon

Pohon adalah bentuk khusus dari graf tak-berarah yang terhubung dan tidak mengandung sirkuit. Misalkan G = (V, E) adalah graf tak-berarah sederhana yang memiliki jumlah simpul n. Maka graf G disebut pohon bila memenuhi syarat: setiap pasang simpul di dalam G terhubung dengan lintasan

tunggal, G terhubung dan memiliki m = n -1 buah sisi, G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit, serta G terhubung dan semua sisinya adalah jembatan.



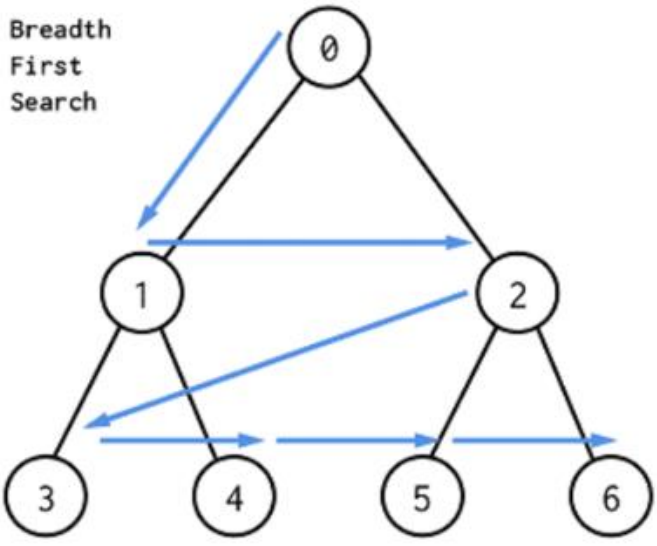
Gambar 1. Contoh pohon
Sumber: Pohon Bagian 1 oleh Rinaldi Munir

C. *Breadth-First Search*

Breadth-First Search adalah algoritma pencarian atau traversal graf yang mengunjungi setiap simpul secara sistematik dengan melakukan pencarian melebar. Pengecekan jalur dilakukan dengan cara membangkitkan dan mengunjungi simpul yang bertetanggaan dengan simpul yang sedang dikunjungi terlebih dahulu, apabila sudah tidak terdapat simpul yang bertetangga maka akan dikunjungi simpul yang merupakan tetangga simpul tadi dan demikian seterusnya.

Algoritma BFS:

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



Gambar 2. Visualisasi algoritma BFS
Sumber: Difference between Depth First Search and Breadth Frist Search oleh Daniel Zaltsman

Struktur data yang diperlukan dalam mengimplementasikan algoritma BFS antara lain:

1. Matriks ketetanggaan A = [a_{ij}] yang berukuran n x n, dengan a_{ij} = 1 jika simpul i dan j bertentanggaan dan a_{ij} = 0 jika tidak.
2. *Queue* untuk menyimpan seluruh simpul yang sudah pernah dikunjungi (dilakukan agar tidak mengunjungi simpul yang sama lebih dari sekali).
3. Tabel boolean yang berisi apakah sebuah simpul i sudah dikunjungi atau belum. Bila sudah maka arrayBoolean[i] = true, apabila belum maka arrayBoolean[i] = false.

Pseudocode algoritma BFS:

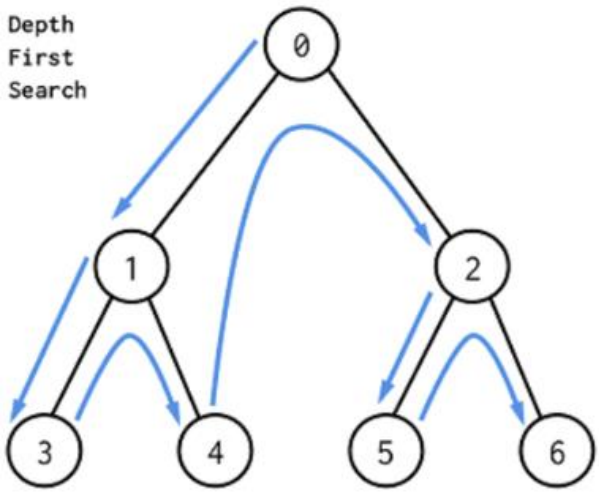
```
procedure BFS (input v: integer)
{input: v merupakan simpul awal yang akan dikunjungi
output: urutan seluruh simpul yang dikunjungi }


Deklarasi
  w: integer
  q: queue

procedure createQueue(input/output q: queue)
{ membuat queue kosong }

procedure pushQueue(input/output q: queue, input v: integer)
{ memasukan v kedalam q pada posisi paling belakang }

procedure popQueue(input/output q: queue, output v: integer)
{ menghapus elemen pertama dari q dan disimpan kedalam variable v }

function emptyQueue(input q: queue) → boolean
{ mengembalikan true apabila q kosong dan false apabila tidak }


Algoritma
  createQueue(q)
  Output(v)
  visited(v) ← true

  pushQueue(q, v)

  while not emptyQueue(q) do
    popQueue(q, v)

    for setiap simpul w yang bertetangga dengan v do
      if not visited[w] then
        Output(w)
        pushQueue(w)
        visited(w) ← true
```

D. Depth-First Search

Depth-First Search adalah algoritma pencarian atau traversal graf yang mengunjungi setiap simpul secara sistematik dengan melakukan pencarian mendalam. Pengecekan jalur dilakukan dengan cara membangkitkan simpul tetangganya dan langsung mengunjungi simpul tetangga tersebut dan demikian seterusnya.

Algoritma DFS:

1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v
3. Ulangi kembali algoritma DFS mulai dari simpul w
4. Keteika mencapai simpul u dimana semua simpul yang bertetangga dengannya telah dikunjungi, pencarian di-*backtrack* ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum pernah dikunjungi
5. Pencarian akan berakhir apabila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapat dari simpul-simpul yang sudah pernah dikunjungi sebelumnya.



Gambar 3. Visualisasi algoritma DFS
Sumber: Difference between Depth First Search and Breadth Frist Search oleh Daniel Zaltsman

Struktur data yang diperlukan dalam mengimplementasikan algoritma DFS antara lain:

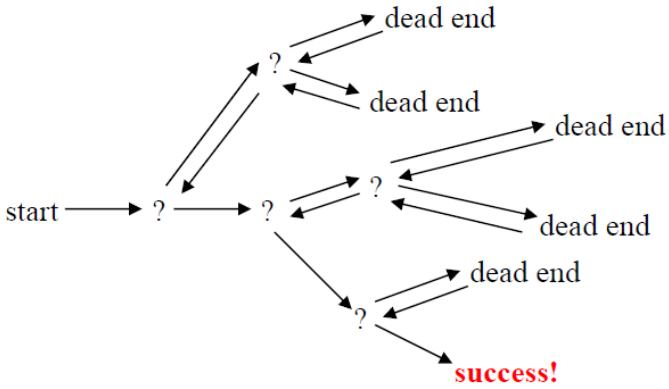
1. Matriks ketetanggaan A = [a_{ij}] yang berukuran n x n, dengan a_{ij} = 1 jika simpul i dan j bertentanggaan dan a_{ij} = 0 jika tidak.
2. Tabel boolean yang berisi apakah sebuah simpul i sudah dikunjungi atau belum. Bila sudah maka visited[i] = true, apabila belum maka visited[i] = false.

Pseudocode algoritma DFS:

```
procedure DFS (input v: integer)
{input: v merupakan simpul awal yang akan dikunjungi
output: urutan seluruh simpul yang dikunjungi }

Deklarasi
  w: integer

Algoritma
  Output(v)
  visited(v) ← true

  for w ← 1 to n do
    if A[v, w] = 1 then
      if not visited[w] then
        DFS(w)
```

E. Algoritma Backtracking

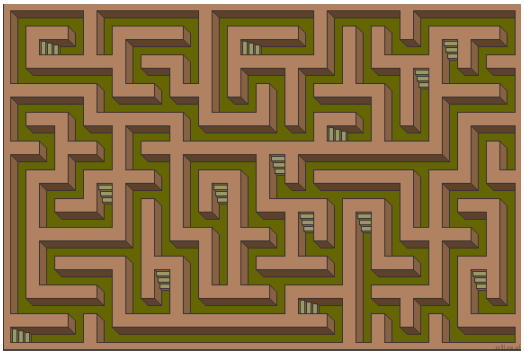
Algoritma *backtracking* atau yang disebut juga dengan algoritma runut-balik pertama kali diperkenalkan oleh D. H. Lehmer pada Tahun 1950. Algoritma *backtracking* merupakan perbaikan dari algoritma *exhaustive search* yang menerapkan pembangkitan status dengan aturan DFS. Cara kerja algoritma *backtracking* adalah menelusuri salah satu kemungkinan solusi secara mendalam hingga mendapatkan solusi atau mencapai titik dimana solusi tidak mungkin ditemukan, maka akan dilakukan runut balik dan melakukan hal yang sama pada kemungkinan solusi lain.



Gambar 4. Visualisasi algoritma *backtracking*
Sumber: BFS dan DFS bagian 2 oleh Rinaldi Munir

F. *Maze*

Maze atau yang disebut juga dengan labirin merupakan kumpulan jalur rumit yang harus diselesaikan untuk mencapai titik goal dari titik awal. Sebuah *Maze* dikatakan sebagai *perfect maze* apabila setiap titik pada maze tersebut hanya dapat dicapai dengan melalui satu jalur saja. Sebuah perfect maze dapat dimodelkan dengan struktur data pohon atau *tree*.

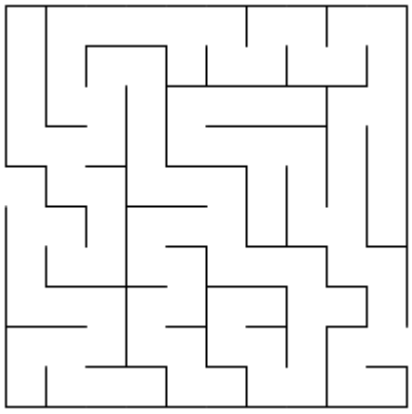


Gambar 5. *Perfect Maze*
Sumber: 'Perfect' Maze Generator oleh cr31

III. PEMBAHASAN

A. *Data Penelitian*

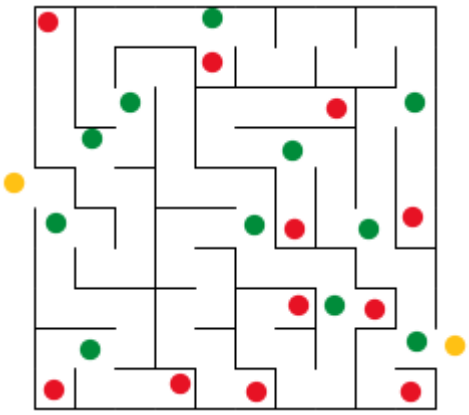
Dalam penulisan makalah ini, penulis akan menggunakan sebuah *maze* / labirin yang diciptakan oleh penulis. Berikut ini *maze* yang akan dicari jalan keluarnya menggunakan algoritma BFS dan DFS.



Gambar 6. Sampel *Perfect Maze*
Sumber: dokumen penulis

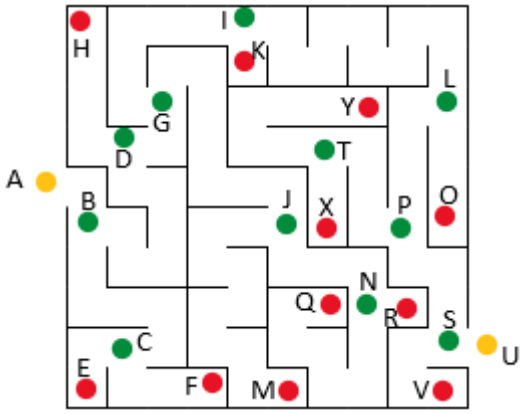
Untuk menyelesaikan persoalan ini maka *maze* tersebut harus diubah kedalam bentuk pohon / *tree* terlebih dahulu. Setiap titik percabangan pada *maze* dapat dijadikan sebagai simpul pada pohon. Terdapat 3 jenis simpul dalam *maze* ini, yaitu simpul mulai atau selesai, simpul titik persimpangan, dan simpul titik buntu.

Dalam representasi persoalan *maze* ini, penulis akan menggunakan warna kuning untuk menandakan simpul mulai atau selesai, warna hijau untuk menandakan simpul titik persimpangan, dan warna merah untuk menandakan simpul titik buntu. Berikut ini gambar pembangkitan simpul dari *maze* yang dijadikan persoalan.

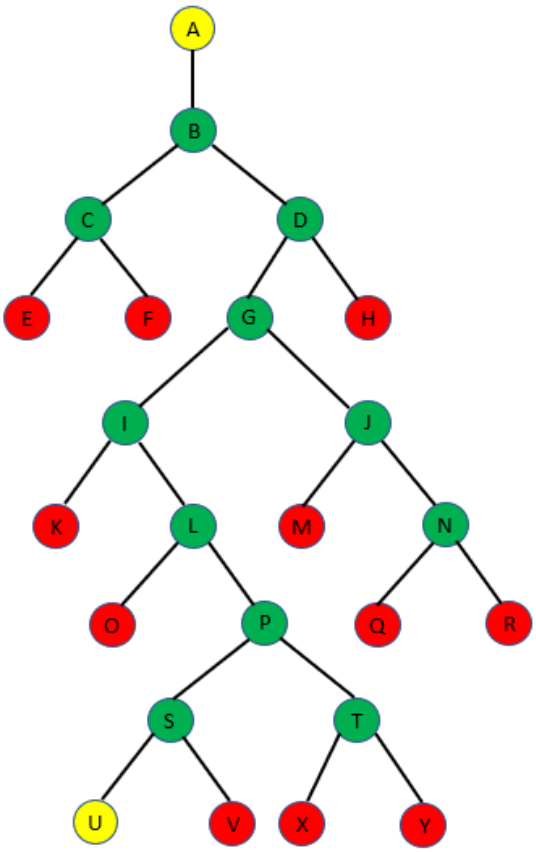


Gambar 7. Simpul dari Sampel *Perfect Maze*
Sumber: dokumen penulis

Simpul – simpul yang telah ditandai pada persoalan *maze* diatas kemudian akan diberikan nama dan direpresentasikan dalam bentuk tree dengan bentuk sebagai berikut.



Gambar 8. Simpul dengan nama dari Sampel *Perfect Maze*
Sumber: dokumen penulis



Gambar 9. Tree dari persoalan sampel
Sumber: dokumen penulis

B. *Aplikasi Algoritma Breadth-First Search Dalam Menyelesaikan Persoalan Maze*

Penyelesaian *maze* dengan algoritma BFS dapat menggunakan pendekatan struktur data *queue* (antrian). Dengan menggunakan struktur data queue, maka setiap simpul yang hidup diperlakukan dengan metode *First In First Out* (FIFO) yang artinya simpul yang duluan masuk akan duluan keluar. Berikut ini tabel pencarian solusi dari persoalan *maze* menggunakan algoritma BFS.

Simpul Expand	Simpul Hidup
A	B_A
B_A	C_{AB}, D_{AB}
C_{AB}	D_{AB}, E_{ABC}, F_{ABC}
D_{AB}	E_{ABC}, F_{ABC}, G_{ABD}, H_{ABD}
E_{ABC}	F_{ABC}, G_{ABD}, H_{ABD}
F_{ABC}	G_{ABD}, H_{ABD}
G_{ABD}	H_{ABD}, I_{ABDG}, J_{ABDG}
H_{ABD}	I_{ABDG}, J_{ABDG}
I_{ABDG}	J_{ABDG}, K_{ABDGI}, L_{ABDGI}
J_{ABDG}	K_{ABDGI}, L_{ABDGI}, M_{ABDGJ}, N_{ABDGJ}
K_{ABDGI}	L_{ABDGI}, M_{ABDGJ}, N_{ABDGJ}
L_{ABDGI}	M_{ABDGJ}, N_{ABDGJ}, O_{ABDGIL},
	P_{ABDGIL}
M_{ABDGJ}	N_{ABDGJ}, O_{ABDGIL}, P_{ABDGIL}
N_{ABDGJ}	O_{ABDGIL}, P_{ABDGIL}, Q_{ABDGJN}, R_{ABDGJN}
O_{ABDGIL}	P_{ABDGIL}, Q_{ABDGJN}, R_{ABDGJN}
P_{ABDGIL}	Q_{ABDGJN}, R_{ABDGJN}, $S_{ABDGILP}$, $T_{ABDGILP}$
Q_{ABDGJN}	R_{ABDGJN}, $S_{ABDGILP}$, $T_{ABDGILP}$
R_{ABDGJN}	$S_{ABDGILP}$, $T_{ABDGILP}$
$S_{ABDGILP}$	$T_{ABDGILP}$, $U_{ABDGILPS}$, $V_{ABDGILPS}$
$T_{ABDGILP}$	$U_{ABDGILPS}$, $V_{ABDGILPS}$, $X_{ABDGILPT}$, $Y_{ABDGILPT}$
$U_{ABDGILPS}$	$V_{ABDGILPS}$, $X_{ABDGILPT}$, $Y_{ABDGILPT}$

Jadi solusi dari persoalan maze tersebut adalah A-B-D-G-I-L-P-S-U, yang dapat diselesaikan oleh algoritma BFS dalam 21 langkah atau 21 pembangkitan simpul.

C. *Aplikasi algoritma Depth-First Search dalam menyelesaikan persoalan maze*

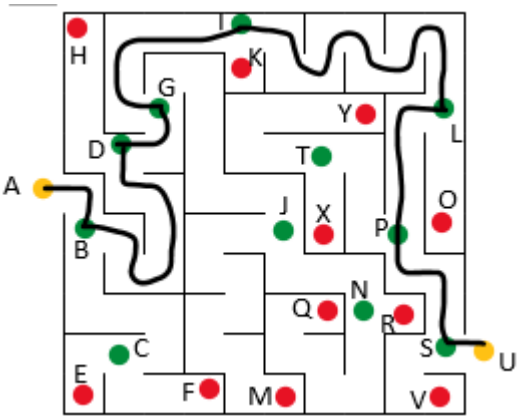
Penyelesaian *maze* dengan algoritma DFS dapat menggunakan pendekatan struktur data *stack* (tumpukan). Dengan menggunakan struktur data *stack*, maka setiap simpul yang hidup diperlakukan dengan metode *Last In First Out* (LIFO) yang artinya simpul yang terakhir masuk akan duluan keluar. Berikut ini tabel pencarian solusi dari persoalan *maze* menggunakan algoritma DFS.

Simpul Expand	Simpul Hidup
A	B_A
B_A	C_{AB}, D_{AB}
C_{AB}	E_{ABC}, F_{ABC}, D_{AB}
E_{ABC}	F_{ABC}, D_{AB}
F_{ABC}	D_{AB}
D_{AB}	G_{ABD}, H_{ABD}
G_{ABD}	I_{ABDG}, J_{ABDG}, H_{ABD}
I_{ABDG}	K_{ABDGI}, L_{ABDGI}, J_{ABDG}, H_{ABD}
K_{ABDGI}	L_{ABDGI}, J_{ABDG}, H_{ABD}
L_{ABDGI}	O_{ABDGI}, P_{ABDGIL}, J_{ABDG}, H_{ABD}
O_{ABDGI}	P_{ABDGIL}, J_{ABDG}, H_{ABD}
P_{ABDGIL}	$S_{ABDGILP}$, $T_{ABDGILP}$, J_{ABDG}, H_{ABD}
$S_{ABDGILP}$	$U_{ABDGILPS}$, $V_{ABDGILPS}$, $T_{ABDGILP}$, J_{ABDG}, H_{ABD}
$U_{ABDGILPS}$	$V_{ABDGILPS}$, $T_{ABDGILP}$, J_{ABDG}, H_{ABD}

Jadi solusi dari persoalan maze tersebut adalah A-B-D-G-I-L-P-S-U, yang dapat diselesaikan oleh algoritma DFS dalam 14 langkah atau 14 pembangkitan simpul.

D. Hasil Penelitian

Berdasarkan hasil penelusuran atau pencarian solusi menggunakana algoritma *Breadth-First Search* dan *Depth-First Search*, solusi dari persoalan *maze* adalah A-B-D-G-I-L-P-S-U.



Gambar 10. Solusi dari persoalan sampel
Sumber: dokumen penulis

Penggunaan algoritma BFS menghasilkan solusi dalam 21 langkah atau 21 pembangkitan simpul sedangkan algoritma DFS dapat menyelesaikan persoalan dalam 14 langkah.

IV. Kesimpulan

Pencarian solusi dari persoalan *maze* dapat dilakukan dengna menggunakan pendekatan mengubah *maze* menjadi sebuah tree dan melakukan pencarian dengan algoritma BFS dan DFS. Dengan menggunakan algoritma BFS dan DFS maka diperoleh solusi dari persoalan *maze* yang disajikan pengguna adalah A-B-D-G-I-L-P-S-U. Algortima BFS membutuhkan 21 langkah atau pembangkitan simpul untuk memperoleh solusi sedangkan DFS dapat menyelesaikan persoalan dalam 14 langkah.

Dalam kasus pencarian solusi pada *maze* akan cenderung lebih cepat menggunakan algoritma DFS karena goal atau jalan keluar dari *maze* biasanya terletak pada ujung yang berlawanan dari jalur masuk, hal ini menandakan bahwa goal node akan ada pada bagian dalam. Ditambah lagi tujuan dari persoalan *maze* adalah membuat bingung orang yang ingin menelusurinya, sehingga goal pasti tidak akan diletakkan dekat dengan starting node. Oleh karena itu, pencarian menggunakan algoritma DFS yang mengutamakan kedalaman akan mendapatkan solusi dalam langkah yang lebih sedikit dibandingkan dengan algoritma BFS yang mencari secara melebar.

V. Ucapan Terima Kasih

Penulis ingin menyampaikan ucapan syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya, penulis diberikan kesehatan dan kekuatan untuk menyelesaikan penulisan makalah ini dengan tepat waktu. Penulis ingin berterima kasih kepada kedua orang tua penulis yang telah mendukun secara moral, doa, dan material. Penulis juga ingin menyampaikan terima kasih kepada semua pihak yang telah mendukung studi dan proses pembelajaran pada mata kuliah IF2211 Strategi Algoritma terutama kepada dosen pengampu mata kuliah ini yaitu, Bapak Dr. Rinaldi Munir atas bimbingan dan ilmu yang telah diberikan selama 1 semester tahun ajaran 2021/2022 ini.

Video Link at Youtube

Berikut ini penulis lampirkan link video penjelasan makalah ini di youtube: https://youtu.be/iQcnmkceplw

Referensi

[1] Munir, Rinaldi Graf Bagian 1. Informatika, Bandung: 2020. Diakses tanggal 19 Mei 2022.
[2] Munir, Rinaldi Graf Bagian 2. Informatika, Bandung: 2020. Diakses tanggal 19 Mei 2022.
[3] Munir, Rinaldi Graf Bagian 3. Informatika, Bandung: 2020. Diakses tanggal 19 Mei 2022.
[4] Munir, Rinaldi Pohon Bagian 1. Informatika, Bandung: 2021. Diakses tanggal 19 Mei 2022.
[5] Munir, Rinaldi Pohon Bagian 2. Informatika, Bandung: 2021. Diakses tanggal 19 Mei 2022.
[6] Munir, Rinaldi Breadth/Depth Firs Search Bagian 1. Informatika, Bandung: 2021. Diakses tanggal 19 Mei 2022.
[7] Munir, Rinaldi Breadth/Depth Firs Search Bagian 2. Informatika, Bandung: 2021. Diakses tanggal 19 Mei 2022.
[8] Zaltsman Daniel Difference between Depth First Search and Breadth First Seacrh. Diakses tanggal 20 Mei 2022

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Mei 2022

Vincent Ho (13520093)