Aplikasi Algoritma *Depth-First Search* dalam Penyelesaian Labirin

Joel Triwira, 13519073
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : triwira.joel@gmail.com

***Abstract*—Labirin merupakan suatu persoalan umum yang sering ditemukan dalam dunia informatika. Biasanya untuk persoalan labirin ini, banyak orang menggunakan algoritma DFS, BFS, A*, dan IDS untuk menemukan jalan keluarnya. Algoritma-algoritma ini memang digunakan untuk permasalahan mencari rute dari titik awal sampai titik tujuan. Akan tetapi, sebagai manusia pasti akan mengalami kesulitan untuk memecahkan masalah labirin bila ia sendiri yang berada dalam labirin tersebut, apalagi bila labirin berukuran sangat luas karena adanya keterbatasan dalam menyimpan ingatan/memori sehingga manusia tidak dapat menerapkan algoritma-algoritma di atas. Maka dari itu, manusia biasanya menggunakan mesin komputer untuk menyelesaikannya. Disini kita akan membahas cara menyelesaikan labirin dengan DFS.**

Keywords—algoritma; DFS; BFS; A*; IDS; labirin; orientasi arah; pohon ruang status; runut balik.

I. PENDAHULUAN

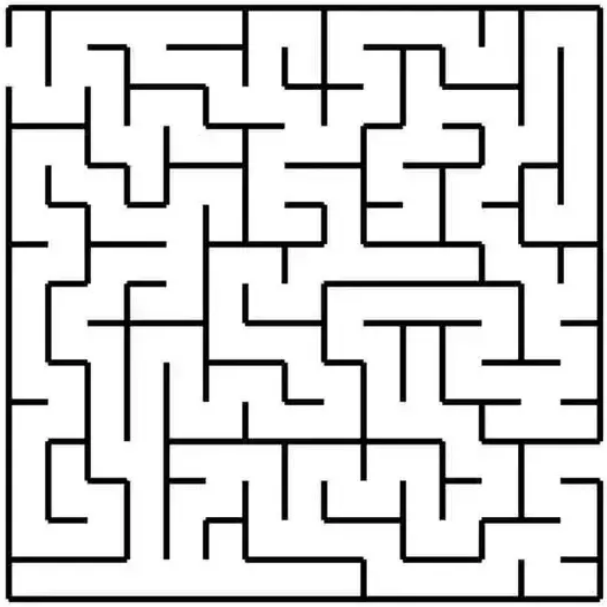
Dalam informatika, persoalan-persoalan yang ada di kehidupan sehari-hari dapat diselesaikan dengan menggunakan komputer dan sebagai teknik informatika, kita dapat harus menyelesaikan masalah-masalah tersebut dengan komputer. Komputer tersebut pastinya sudah diprogram atau dipasang perangkat lunak yang dirancang khusus untuk melakukan pemecahan persoalan tersebut. Sebagai contoh, untuk masalah komunikasi terdapat aplikasi *whatsapp*. Aplikasi *whatsapp* digunakan untuk berkomunikasi jarak jauh. Ini bekerja sebagai pengganti SMS dan telepon menggunakan pulsa dan aplikasi ini bekerja dengan menggunakan internet. Namun banyak aplikasi yang mempunyai fungsi mirip dengan *whatsapp* sehingga dipaksa aplikasi-aplikasi dipaksa untuk berkembang lebih. Sebagai contoh, *whatsapp* sekarang aplikasi memiliki fitur mengirim stiker, fitur *story* yang mengupload cerita, dll.

Dewasa Semakin berkembangnya era komputer, semakin banyak masalah yang muncul. Semakin banyak masalah yang muncul, semakin banyak perangkat lunak yang dikembangkan untuk memecahkan masalah-masalah tersebut. Masalah-masalah ini bukan hanya untuk masalah sehari-hari, namun masalah dapat berupa sebagai jembatan melatih cara berpikir. Masalah yang melatih cara berpikir biasa disebut *puzzle*. Biasanya masalah seperti ini mengandung unsur matematika. Contoh-contoh masalah yang melatih cara pikir adalah sudoku, labirin, dll. Kita akah bahas lebih lanjut tentang labirin.

Labirin menurut Kamus Besar Bahasa Indonesia(KBBI) didefinisikan :

1 tempat yang penuh dengan jalan dan lorong yang berliku-liku dan simpang siur; **2** sesuatu yang sangat rumit dan berbelit-belit (tentang susunan, aturan, dan sebagainya); **3** sistem rongga atau saluran yang berhubungan;

Jadi, labirin adalah suatu tempat yang terdiri dari jalan yang berliku-liku dengan jalan masuk dan keluar yang relatif sedikit.



Gambar 1.1 *Labirin*

Gambar 1.1 merupakan contoh dari gambar labirin berbentuk 2 dimensi. Dapat dilihat bahwa dalam labirin diatas hanya terdapat 2 jalan masuk/keluar. Biasanya dalam labirin seperti ini, kita diminta untuk mencari jalan masuk dan keluar. Untuk menyelesaikan persoalan ini, algoritma yang dipakai adalah pencarian BFS dan algoritma pencarian DFS. Kedua algoritma ini dikhususkan untuk mencari rute dari titik awal sampai titik tujuan dan ini adalah salah satu fungsinya. Perbedaan dari algoritma BFS dan DFS adalah dalam cara

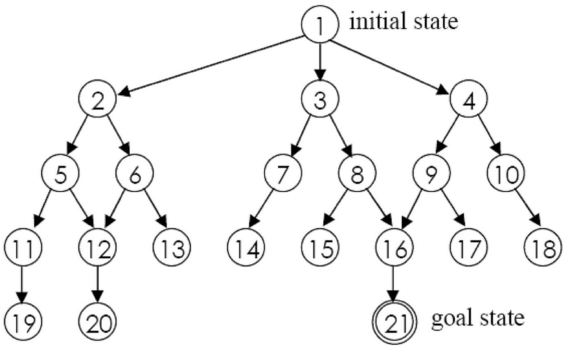
pencariannya. BFS menggunakan cara yang lebih memakan memori, sedangkan DFS menggunakan cara yang lebih memakan waktu.

Untuk penjelasan lebih jelas, makalah ini akan membahas tentang aplikasi algoritma DFS untuk menyelesaikan persoalan labirin.

II. Dasar Teori

A. *Depth-First-Search (DFS)*

Algoritma *Depth-First-Search*, atau lebih dikenal dengan DFS, merupakan algoritma traversal graf yang melakukan penelusuran simpul dengan pendekatan mendalam. Algoritma DFS *Search* memiliki prioritas untuk mengunjungi simpul sampai *level* terdalam terlebih dahulu. Kemudian jika ditemukan jalan buntu (tidak ada lagi simpul yang bertetangga), algoritma akan memeriksa simpul sebelumnya yang sudah dikunjungi dan masih bertetangga dengan simpul lain yang belum dikunjungi dan menelusuri simpul tersebut. Dengan kata lain, simpul cabang atau anak yang terlebih dahulu dikunjungi. Sebagai ilustrasinya dapat dilihat gambar 2.1.



Gambar 2.1 Algoritma pencarian DFS

Depth First Search memiliki kelebihan di antaranya adalah cepat mencapai kedalaman ruang pencarian. Jika diketahui bahwa lintasan solusi permasalahan akan panjang maka *Depth First Search* tidak akan memboroskan waktu untuk melakukan sejumlah besar keadaan dangkal dalam permasalahan graf. *Depth First Search* jauh lebih efisien untuk ruang pencarian dengan banyak cabang karena tidak perlu mengeksekusi semua simpul pada suatu *level* tertentu pada daftar *open.* Selain itu, *Depth First Search* memerlukan memori yang relatif kecil karena banyak *node* pada lintasan yang aktif saja yang disimpan.

Selain kelebihan, *Depth First Search* juga memiliki kelemahan di antaranya adalah memungkinkan tidak ditemukannya tujuan yang diharapkan dan hanya akan mendapatkan satu solusi pada setiap pencarian.

Berikut ini adalah algoritma *Depth First Search* dalam bentuk *pseudocode* :

```
DFS(G, u)
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)

init() {
    For each u ∈ G
        u.visited = false
     For each u ∈ G
       DFS(G, u)
}
```

Penentuan teknik pencarian yang sesuai atau yang tepat untuk sebuah kasus khusus penganalisaan suatu ruang permasalahan, menjadi sangat penting dan biasanya dengan melakukan konsultasi dengan para pakar di bidangnya untuk mendapatkan dan mengetahui tingkah laku ruang permasalahan tersebut.

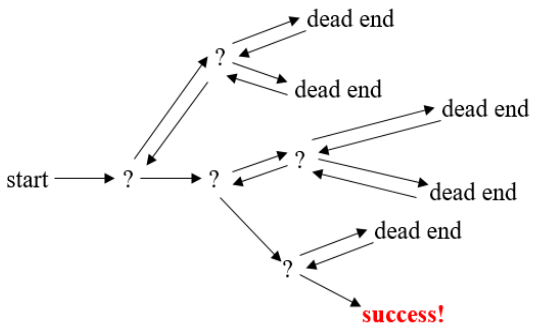
B. *Algoritma Backtracking*

Algoritma backtrack pertama kali diperkenalkan oleh D.H. Lehmer pada tahun1950. Dalam perkembangannya beberapa ahli seperti Rwalker, Golomb, dan Baumert menyajikan uraian umum tentang backtrack dan penerapannya dalam berbagai persoalan dan aplikasi.

Algoritma backtracking (runut balik) merupakan salah satu metode pemecahan masalah yang termasuk dalam strategi yang berbasis pencarian pada ruang status. Algoritma backtracking bekerja secara rekursif dan melakukan pencarian solusi persoalan secara sistematis pada semua kemungkinan solusi yang ada. Oleh karena algoritma ini berbasis pada algoritma Depth-First Search (DFS), maka pencarian solusi dilakukan dengan menelusuri suatu struktur berbentuk pohon berakar secara preorder. Algoritma DFS merupakan algoritma pencarian secara mendalam. Cara kerja algoritma ini adalah menelusuri salah satu kemungkinan yang ada hingga akar-akar maksimal, baru setelah itu menelusuri kemungkinan yang lain hingga akar-akar maksimal juga. Proses ini dicirikan dengan ekspansi simpul terdalam lebih dahulu sampai tidak ditemukan lagi suksesor dari suatu simpul.

```
while belum sampai pada tujuan do
  if terdapat arah yang benar sedemikian sehingga kita belum pernah
     berpindah ke sel pada arah tersebut
  then
     pindah satu langkah ke arah tersebut
  else
     backtrack langkah sampai terdapat arah seperti yang disebutkan
     di atas
  endif
endwhile
```

Gambar 2.2 Gambaran umum algoritma backtracking



Gambar 2.3 Ilustrasi backtracking

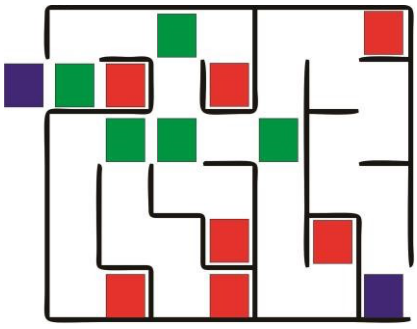
Prinsip dasar algoritma Backtracking adalah mencoba semua kemungkinan solusi yang ada. Perbedaan utamanya adalah pada konsep dasarnya,yaitu pada backtracking semua solusi dibuat dalam bentuk pohon solusi (tree), dan kemudian pohon tersebut akan ditelusuri secara DFS (Depth First Search) sehingga ditemukan solusi terbaik yang diinginkan.

III. Pembahasan

Before Disini kita akan membahas pencarian jalan keluar sebuah labirin dengan algoritma DFS.

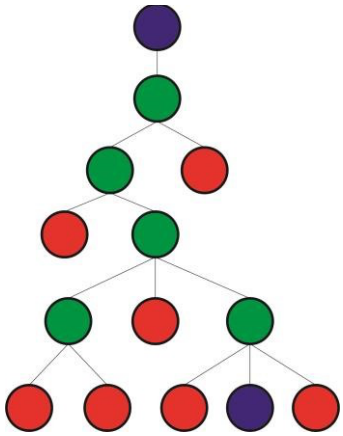
A. Merepresentasi Labirin dalam bentuk Graf

Pertama, untuk menyelesaikan persoalan ini, kita butuh merubah labirin ke dalam bentuk graf untuk dapat dikerjakan. Simpulnya ialah merupakan persimpangan dan sisinya ialah jalan penghubung persimpangan satu dengan lainnya, termasuk dengan jalan buntu. Graf labirin ini memilii 3 jenis simpul, yaitu simpul pintu masuk dan keluar, simpul persimpangan, simpul jalan buntu.



Gambar 3.1 Contoh gambar labirin dengan 3 jenis simpul yang berbeda warna

Pada gambar 3.1, dapat dilihat terdapat 3 warna yang berbeda untuk menggambarkan 3 jenis simpul yang berbeda. Warna ungu melambangkan jalan masuk dan keluar, warna hijau melambangkan persimpangan, warna merah melambangkan jalan buntu. Lalu kita dapat merubahnya menjadi graf.



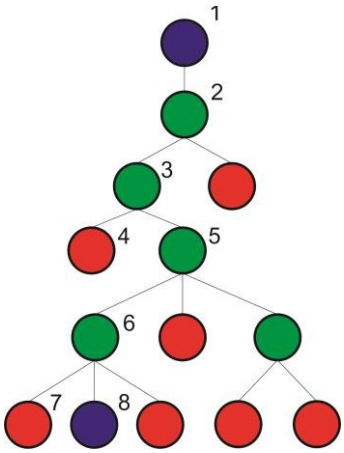
Gambar 3.2 Graf dari bentuk labirin

Terlihat pada gambar 3.2. bahwa simpul berwarna hijau akan membangkitkan simpul baru, sedangkan simpul berwarna merah tidak membangkitkan simpul lagi. Kasus untuk simpul biru berbeda, yakni simpul yang merupakan status awal pasti akan membangkitkan satu simpul saja, sedangkan simpul yang merupakan status solusi tidak akan membangkitkan apapun. Graf pada Ilustrasi 3.2. juga menggambarkan pohon ruang status dimana operator transformasinya ialah arah belokan yang diambil ketika bertemu simpangan, antara lain kiri, lurus, dan kanan.

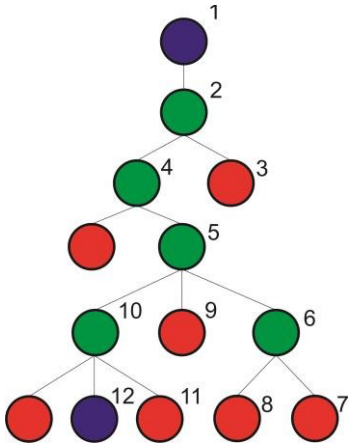
B. Prioritas Arah

Dalam pencarian DFS, dibutuhkan suatu patokan dalam mencari jalur yang terdalam. Misalnya jika dalam bentuk pohon, jalur selanjutnya yang akan dicari akan selalu yang berada di sebelah kiri simpul. Demikian juga dalam labirin. Setelah merubah kedalam bentuk graf, harus ditentukan simpul selanjutnya agar ada patokan prioritas.

Berikut adalah 2 cara menyelesaikan labirin dengan algoritma DFS dengan prioritas arah yang berbeda.



Gambar 3.3 Penyelesaian labirin dengan algoritma DFS yang memiliki prioritas arah ke kiri



Gambar 3.4 Penyelesaian labirin dengan algoritma DFS yang memiliki prioritas arah ke kanan

Dapat dilihat bahwa prioritas arah sangat penting dalam pencarian DFS karena dapat mempersingkat waktu yang ditelusuri jika sudah diketahui arah kemungkinan yang lebih dekat dengan jalan keluar.

Adapun algoritma pengunjungan simpul dan pembangkitan simpul sesuai dengan prioritas arah, misalnya kiri, adalah sebagai berikut.

- Apabila simpul yang dikunjungi ialah simpul jalan buntu, STOP.
- Apabila simpul yang dikunjungi ialah simpul persimpangan, periksa bentuk simpangannya dan jalan baru yang bisa dilalui oleh objek. Setidaknya dua di antara tiga tahap dibawah ini pasti akan dikerjakan.
- Bila dari arah masuk persimpangan terdapat belokan ke kiri, bangkitkan/kunjungi simpul yang dihubungkan dengan jalan tersebut. Ulangi dari langkah 1 dengan posisi sekarang ialah simpul yang baru dibangkitkan atau dikunjungi. Bila tidak ada belokan ke kiri, lanjutkan ke tahap berikutnya.
- Bila dari arah masuk persimpangan terdapat jalan lurus, bangkitkan/kunjungi simpul yang dihubungkan dengan jalan lurus tersebut. Ulangi dari langkah 1 dengan posisi sekarang ialah simpul yang baru dibangkitkan atau dikunjungi. Bila tidak ada jalan lurus, lanjutkan ke tahap berikutnya.
- Bila dari arah masuk persimpangan terdapat belokan ke kanan, bangkitkan/kunjungi simpul yang dihubungkan dengan belokan ke kanan. Ulangi dari langkah 1 dengan posisi sekarang ialah simpul yang baru dibangkitkan atau dikunjungi. Bila tidak ada belokan ke kanan, lanjutkan ke tahap berikutnya.
- Runut balik ke simpul persimpangan sebelumnya.
- Bila simpul sekarang ialah simpul awal, STOP.

```
function SolveMaze(input M : labirin)→boolean
{ true jika pilihan mengarah ke solusi }

Deklarasi
   arah : integer  { up = 1, down, 2, left = 3, right = 4
}

Algoritma:
   if pilihan arah merupakan solusi then
     return true
   else
     for tiap arah gerakan (lurus, kiri, kanan) do
        move(M, arah)   { pindah satu langkah (satu sel)
                          sesuai arah  tersebut }
        if SolveMaze(M) then
          return true
        else
          unmove(M, arah)   { backtrack }
        endif
     endfor
     return false     { semua arah sudah dicoba, tetapi
                        tetap buntu,  maka
                        kesimpulannya: bukan solusi }
   endif
```

Gambar 3.5 Pseudocode untuk persoalan labirin

Berikut adalah contoh aplikasi menggunakan python.

```
import sys


class MazeSolver:
    def __init__(self, nama_file):
        self.labirin = []
        self.lebar = 0
        self.panjang = 0
        self.solusi = []
        self.dilewati =[]
        self.start = []
        self.finish = []

        self.baca_maze(nama_file)
        self.solve()

    def baca_maze(self, nama_file):
        f = open(nama_file, 'r')
        for baris in f:
            lst = []
            for kolom in baris:
                if kolom == '#':
                    lst.append(0)
                elif kolom == ' ':
                    lst.append(1)
            self.labirin.append(lst)
```

```
            self.panjang                    =
len(self.labirin)
            self.lebar                      =
len(self.labirin[0])
            self.cari_start()
            self.cari_finish()


        def cari_start(self):
            i = 0
            ketemu = False
            # Cari Di Sumbu X
            for x in self.labirin[0]:
                if x == 1:
                    ketemu = True
                    self.start = [0, i]
                    break
                i += 1


            # Cari di Sumbu Y
            i = 0
            for     y     in     range(0,
len(self.labirin)):
                if  self.labirin[y][0]  ==
1:
                    self.start =  [i, 0]
                    break
                i += 1


        def cari_finish(self):
            i = 0
            ketemu = False
            # Cari Di Sumbu X
            for x in self.labirin[-1]:
                if x == 1:
                    ketemu = True
                    self.finish          =
[len(self.labirin) - 1, i]
                    break
                i += 1
```

```
            # Cari di Sumbu Y
            i = 0
            for     y     in     range(0,
len(self.labirin)):
                if
self.labirin[y][len(self.labirin[y])    -
1] == 1:
                    self.finish    =    [y,
len(self.labirin[y]) - 1]
                    break
                i += 1


        def cari_jalan(self, x, y):
            if [y, x] == self.finish:
                self.solusi.append([y, x])
                self.tampil_solusi()
            else:
                if (y - 1 > 0) and (([y -
1,   x]   not   in   self.dilewati)   and
(self.labirin[y - 1][x] > 0)): # Ke Atas
                    self.solusi.append([y,
x])

self.dilewati.append([y - 1, x])
                    self.cari_jalan(x, y -
1)


                elif (y + 1 < self.panjang)
and (([y + 1, x] not in self.dilewati)
and (self.labirin[y + 1][x] > 0)): # Ke
Bawah
                    self.solusi.append([y,
x])

self.dilewati.append([y + 1, x])
                    self.cari_jalan(x, y +
1)


                elif (x + 1 < self.lebar)
and (([y, x + 1] not in self.dilewati)
and (self.labirin[y][x + 1] > 0)): # Ke
Kanan
                    self.solusi.append([y,
x])

self.dilewati.append([y, x + 1])
```

```
                    self.cari_jalan(x + 1,
y)


                elif (x - 1 > 0) and (([y,
x - 1] not in self.dilewati) and
(self.labirin[y][x - 1] > 0)): # Ke Kiri
                    self.solusi.append([y,
x])

self.dilewati.append([y, x - 1])
                    self.cari_jalan(x - 1,
y)


                else:
                    if len(self.solusi) >
0:
                        sebelum =
self.solusi.pop()

self.cari_jalan(sebelum[1], sebelum[0])
# Backtrack
                    else:
                        print "Solusi
Tidak Ditemukan"


        def solve(self):
            start = self.start
            self.cari_jalan(start[1],
start[0])


        def tampil_solusi(self):
            for br in range(0,
self.panjang):
                for kl in range(0,
self.lebar):
                    if [br, kl] in
self.solusi:

sys.stdout.write(".")
                    else:
                        if
self.labirin[br][kl] == 0:

sys.stdout.write("#")
                        elif
self.labirin[br][kl] == 1:

sys.stdout.write(" ")
                sys.stdout.write("\n")


s = MazeSolver('maze.txt')
```

IV. Kesimpulan

Algoritma DFS dapat banyak digunakan dalam masalah sehari-hari untuk membantu manusia dan objek lain. Terutama dalam makalah ini membantu mencari rute jalan keluar dalam jalan berliku-liku. Algoritma DFS ini juga dapat digunakan untuk mencari jalan tercepat di peta dunia. Efisiensi waktu sangat bergantung terhadap bentuk labirin dan prioritas arah yang diambil oleh objek saat memulai memecahkan persoalan ini. Semakin banyak simpul yang ada akan semakin lama algoritma ini berjalan. Arah juga menentukan, jika memang lebih dekat dengan satu arah, tentu akan lebih cepat jika memilih arah yang benar. Efisiensi ruang untuk DFS sudah sangat efisien karena tidak membutuhkan memori banyak. Setiap bertemu dengan jalan buntu, algoritma DFS akan melakukan DFS sehingga akan melepas memori. Dalam kasus ini, labirin dapat diselesaikan dengan dirubah menjadi graf berbentuk pohon, lalu mengaplikasikan algoritma DFS terhadap graf dengan memasukkan simpul awal dan simpul tujuan akhir.

V. Ucapan Terima Kasih

Pertama-tama saya mengucapkan syukur dan terima kasih yang sebesar-besarnya kepada Tuhan Yang Maha Esa karena rahmat dan berkat karunia-Nya yang selalu menyertai penulis sehingga makalah ini dapat selesai dirancang dan ditulis. Penulis juga berterimakasih banyak kepada keluarga saya terutama kedua orang tua saya yang selalu mendukung saya dalam pengerjaan makalah ini melalui doa dan bantuan dalam bentuk moral dan material. Dan terakhir, penulis ingin berterimakasih Ibu Nur Ulfa maulidevi selaku dosen mata kuliah Strategi Algoritma yang telah mengajar banyak hal mengenai algoritma-algoritma.

Video Link at Youtube *(Heading 5)*

Include link of your video on YouTube in this section.

Daftar Pustaka

[1] *How to Find Your Way Through A Maze*. (t.thn.). Dipetik Mei 7, 2016, dari wikiHow: http://www.wikihow.com/Find-Your-WayThough-a-Maze

[2] Kleinberg, J., & Tardos, E. (2006). *Algorithm Design.* Massachussets: Pearson Education, Inc.

[3] Munir, Rinaldi. 2021. “Breath/Depth First Search (Bagian 1)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf, diakses pada tanggal 10 Mei 2021

[4] Munir, Rinaldi. 2021. “Algoritma Runut-Balik (Bagian 1)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf, diakses pada tanggal 10 Mei 2021

[5] Munir, Rinaldi. 2021. “Breath/Depth First Search (Bagian 2)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf, diakses pada tanggal 10 Mei 2021

[6] Munir, Rinaldi. 2021. “Algoritma Runut-Balik (Bagian 2)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian2.pdf, diakses pada tanggal 10 Mei 2021

[7] Nafi’ah, Dzurrotun, dkk. Analisis Penerapan Algoritma Backtrack dalam Pencarian Solusi Game Maze (Labirin). Jurusan Teknik Informatika, Sekolah Tinggi Teknologi Telkom

[8] Teneng, dkk. Analisis Penerapan Algoritma Backtrack dalam Pencarian Solusi Game Maze (Labirin). Program Studi Teknik Informatika Universitas Kristen Duta Wacana Yogyakarta

[9] Fauzan, Irsyad. 2017. “Maze Solver Menggunakan Algoritma Backtracking”, https://irsyadf.my.id/maze-solver-menggunakan-algoritma-backtracking-47249d1542af, diakses pada tanggal 11 Mei 2021

[10]

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, 9 Mei 2021

Joel Triwira