Penerapan Algoritma BFS dalam Penyelesaian Permainan The Labyrinth

Mochammad Fatchur Rochman 13519009
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519009@std.stei.itb.ac.id

***Abstract*—Dalam menyelesaikan persoalan penelesuran sebuah graf dengan tidak ada informasi tambahan (Uniformed/ Blind search), salah satu pendekatan yang digunakan adalah dengan penelusuran secara melebar yaitu dengan algoritma BFS (Breadth First Search). Pada makalah ini akan ditunjukkan bagaimana algoritma BFS menyelesaikan permainan The Labyrinth.**

Kata kunci—BFS, Uniformed/Blind search, The Labyrinth

I. PENDAHULUAN

The Labyrinth adalah salah satu permainan yang terdapat di situs *Codingame*, permainan ini merupakan puzzle game tentang penelurusan sebuah labirin 2D dengan cara yang optimal.

Link: https://www.codingame.com/ide/puzzle/the-labyrinth

Permainan *The Labirinth* bercerita Kirk yang sedang melakukan penelusuran di sebuah labirin menggunakan *spaceship*-nya untuk menemukan *control room* kemudian kembali lagi ke posisi awal tempat ia mulai berangkat dalam waktu tertentu.

Gambar 1. Gameplay Permainan

(Sumber : Dokumentasi Penulis)

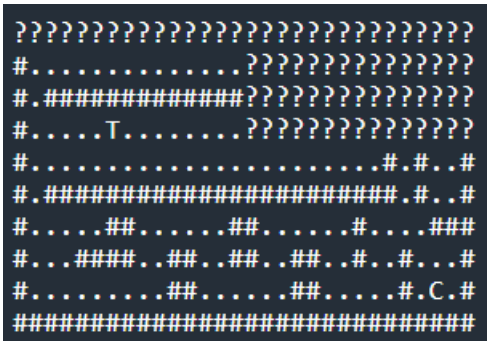
Permainan ini memiliki struktur labirin yang berbentuk persegi Panjang yang tersusun atas sel-sel. Dalam permaianan ini kita tidak diberikan informasi tambahan mengenai letak conrol room tersebut, itulah mengapa penelusuran labirin dalam permainan ini merupakan *uniformed/blind search*.

Permaianan ini memiliki aturan Kirk (karakter dalam game ini) hanya dapat melakukan pergerakan *UP* (atas), *DOWN* (bawah), *LEFT* (kiri), *atau RIGHT* (kanan). Kirk dengan *tricorder*-nya hanya dapat memindai kotak selebar 5 sel dengan dia yang menjadi pusatnya. Setelah ia berhasil mencapai *control room*, akan ada waktu hitung mundur yang harus kita gunakan untuk Kembali ke posisi awal kita berangkat sebelum waktu hitung mundur tersebut habis. Kirk akan gagal dalam misinya bila :

- Spaceship milik Kirk habis bensin-nya. Kirk akan memiliki bensin yang cukup untuk melakuakan 1200 gerakan.
- Kirk tidak berhasil mencapai posisi awal tempat awal ia berangkat, sebelum waktu hitung mundur habis. Waktu hitung mundur dihung tepat setelah Kirk mencapai *control room*.
- Kirk menyenntuh tembok atau tanah.
- Kirk dikatakan berhasil dalam misinya jika ia berhasil mencapai *control room* dan kembali ke posisi awal sebelum waktu hitung mundur habis.

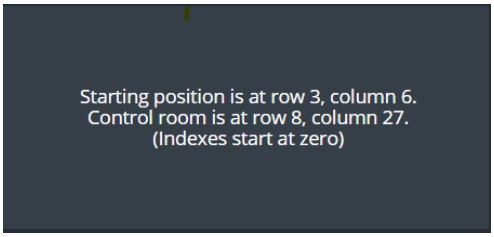
Labirin dalam permainan ini dalam format ASCII yang disediakan oleh input, karakter '#' merepresentasikan tembok, karakter '.' Merepresentasikan ruang hampa, karakter 'T' merepresentasikan posisi awal Kirk berangkat. karakter 'C' merepresentasikan letak *control room* dan karakter '?' merepresentasikan sel yang belum kita pindai/identifikasi.

Contoh :



Gambar 2. Contoh Format Labirin

(Sumber : Dokumentasi Penulis)



Gambar 3. Keterangan Format Labirin

(Sumber : Dokumentasi Penulis)

II. DASAR TEORI

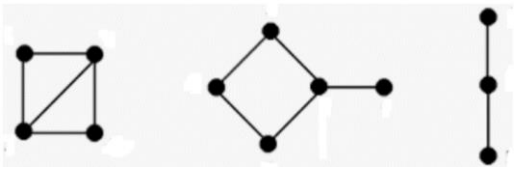
A. *Graf*

Graf dalam matematika dan informatika adalah himpunan dari objek-objek yang dinamakan titik, simpul, atau sudut (Vertex) yang dihubungkan oleh penghubung yang dinamakan garis atau sisi (Edge). Secara formal, graf didefinisikan sebagai G = (V,E) dengan G adalah graf, V adalah sebuah himpunan yang elemennya dinamakan sudut atau simpul, V = {v1,v2,v3,….,vn}. E adalah himpunan sebuah dari pasangan-pasangan sudut yang terpisah, yang dinamakan sisi atau garis, E = {e1,e2,e3,….,en}.

Berdasarkan ada tidaknya gelang atau sisi-ganda pada graf, graf dibedakan menjadi dua jenis:

1. Graf sederhana (simple graph)

   Graf yang didalamnya tidak mengandung gelang maupun sisi ganda.



Gambar 4. Graf sederhana

(Sumber : PPT Rinaldi Munir/IF2120/ Graf bag.1)

2. Graf tak-sederhana (unsimple-graph)

   Graf yang didalamnya mengandung sisi ganda atau gelang.

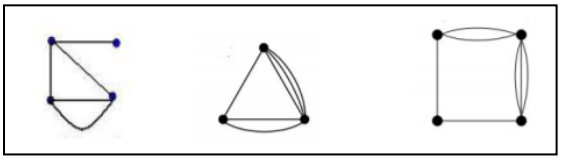
Gambar 5. Graf tak-sederhana

(Sumber : PPT Rinaldi Munir/IF2120/ Graf bag.1)

Graf tak-sederhana dibedakan lagi menjadi:

1. Graf ganda (multi-graph)

   Graf yang mengandung sisi ganda.

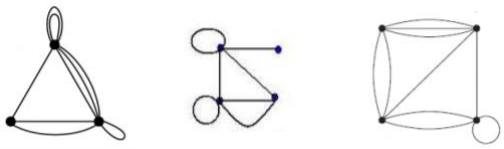


Gambar 6. Graf ganda

(Sumber : PPT Rinaldi Munir/IF2120/ Graf bag.1)

2. Graf semu (psedo-graph)

   Graf yang mengandung sisi gelang.



Gambar 7. Graf semu

(Sumber : PPT Rinaldi Munir/IF2120/ Graf bag.1)

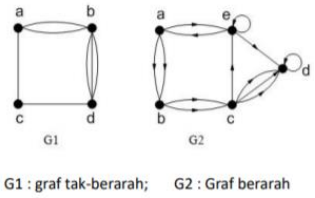
Berdasarkan orientasi arah pada sisi, graf dibedakan atas 2 jenis :

1. Graf tak-berarah (undirected graph)

   Graf yang sisinya tidak mempunyai orientasi arah.

2. Graf berarah (directed graph)

   Graf yang setiap sisinya diberikan orientasi arah.



Gambar 8. Graf tak berarah dan Graf berarah

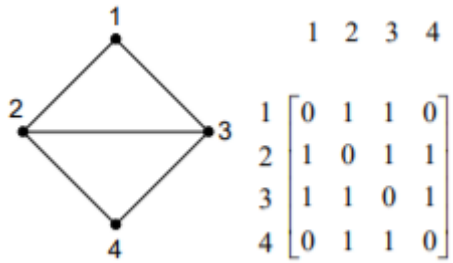
(Sumber : PPT Rinaldi Munir/IF2120/ Graf bag.1)

Dalam praktiknya di pemrograman cara merepesentasikan graf ada tiga jenis:

1. Matriks ketetanggan

   Pada matriks ketetanggan, baris dan kolom matriks akan berisi 1 dan 0 untuk graf tak berbobot. Untuk

setiap A = [a_{ij}], jika simpul i dan j betetangga maka a_{ij} akan bernilai 1, sedangkan jika simpul i dan j tidak bertetangga maka a_{ij} akan bernilai 0.



Gambar 6. Representasi matriks ketetanggaan

(Sumber : PPT Rinaldi Munir/IF2120/ Graf bag.2)

2. Adjacency List

Pada adjacency list, keterhubungan antar simpul didefinisikan dengan menggunakan linked list. Setiap simpul memiliki list dari simpul-simpul yang terhubung dengan simpul tersebut.

Contoh:

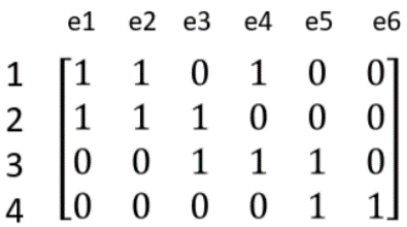
Simpul	Simpul Tetangga
1	2, 3
2	1, 3, 4
3	1, 2, 4
4	2, 3

Gambar 7. Representasi Adjacency list

(Sumber : PPT Rinaldi Munir/IF2120/ Graf bag.2)

3. Matriks Bersisian

Pada matriks bersisian, baris dan kolom matriks berisi 1 dan 0 untuk graf tak berbobot, Untuk setiap A = [a_{ij}], jika simpul i dan j bertetangga maka a_{ij} akan bernilai 1, jika simpul i bersisian dengan sisi j, dan a_{ij} bernilai 0 jika simpul i tidak bersisian dengan sisi j.



Gambar 8. Reperesentasi matriks bersisian

(Sumber : PPT Rinaldi Munir/IF2120/ Graf bag.2)

B. Algoritma Pencarian Graf

Algoritma Pencarian Graf adalah algoritma yang menyelesaikan masalah graf dengan mengevaluasi solusi dari masalah-masalah tersebut. Algoritma pencarian dapat dikategorikan menjadi dua kategori yaitu pencarian solusi tanpa informasi (*Uniformed/blind Search*) dan pencarian solusi dengan informasi (*Informed Search*).

1. Pencarian Solusi Tanpa Informasi (*Uniformed/blind Search*)

   Pencarian ini tidak menggunakan informasi tambahan mengenai sifat alami dari permasalahan sehingga dapat diimplementasikan secara umum. Namun pencarian ini memiliki ruang lingkup solusi yang lebih luas, sehingga membutuhkan waktu yang cukup lama untuk menemukan solusi. Yang termasuk dalam pencarian tanpa informasi : BFS, DFS, IDS, DLS, dan UCS.

2. PencarianSolusi dengan Informasi (*Informed Search*)

   Pencarian ini berbasis heuristik dalam menyelesaikan maslah. Dengan adanya heuristic pencarian lebih terarah dan waktu yang dibutuhkan untuk menyelesaikan masalah jauh lebih cepat. Yang termasuk dalam pencarian dengan informasi tambahan: Best First Seach dan A*.

Dalam merepresentasikan graf pada proses pencarian, terdapat dua pendekatan:

1. Graf statis

   Graf yang sudah terbentuk sebelum proses pencarian dilakukan, graf direpresentasikan sebagai struktur data.

2. Graf dinamis

   Graf yang terbentuk saat proses pencarian dilakukan, graf tidak tersedia sebelum pencarian, graf dibagun selama pencarian solusi.

C. Algorima Breadth First Search

Algoritma traversal graf yang mengunjungi simpul dengan cara yang sistematik secara melebar. Untuk kompleksitas waktu-nya O(|V|+|E|), karena setiap simpul dan sisi dikunjungi dalam kasus terburuknya.

Algoritma:

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

```
procedure BFS(G, root) is
    let Q be a queue
    label root as discovered
    Q.enqueue(root)
    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
            if w is not labeled as discovered then
                label w as discovered
                Q.enqueue(w)
```

Gambar 9. Pseudocode Algoritma BFS
(Sumber: https://en.wikipedia.org/)

III. Implementasi Algoritma BFS

A. *Struktur Data dan Inisialisasi*

Dalam makalah ini bahasa pemrograman yang akan digunakan untuk mengaplikasikan algoritma penyelesaian adalah bahasa Python3.x.

Representasi graf yang akan digunakan adalah graf dinamis, karena selama melakukan penelusuran dari pencarian masih harus diidentifikasi terlebih dahulu (dari di-game diberikan input yang harus diidentifikasi dulu) apakah simpul pada map labirin tersebut dapat diakses (yaitu '.' = ruang hampa, 'T'= posisi awal, 'C' = control room) atau tidak (yaitu '#' yaitu tembok).

Dalam implementasi algoritma BFS digunakan struktur data *queue*/antrian.

```
# preparation
queue = []
colour = []
distance = []
parents = []
for row in range(len(game_map)):
    colour.append([])
    distance.append([])
    parents.append([])
    for collumn in range(len(game_map[row])):
        colour[row].append(0)
        distance[row].append(INFINITY)
        parents[row].append(None)

colour[start[0]][start[1]] = 1
distance[start[0]][start[1]] = 0

queue.append(start)
```

Gambar 10. Inisialisasi untuk Algoritma BFS

(Sumber : Hubbert Jarrosz, 2016)

Inisialisasi array kosong untuk menampung queue/anrian, colour (untuk menampung simpul yang sudah diidentifikasi/sudah dijelajahi), distance untuk menampung jarak dari antar simpul, dan parents yang menampung parent dari suatu simpul yang berada di suatu koordinat. Untuk setiap game_map yang diinputkan oleh sistem game, inisialisasi setiap baris pada array colour dengan 0 atau False, setiap baris array distance dengan INFINITY yang dalam hal ini didefinisikan sebagai float('inf') dalam Python3, dan setiap baris array parents dengan None. Kemudian pada saat akan

B. *Strategi yang digunakan untuk menyelesaikan Permainan The Labyrinth*

Strategi BFS yang digunakan untuk menyelesaikan permainanan ini yaitu dengan melakukan penelusuran untuk setiap area game_map yaitu area yang untuk setiap saat itu telah terscan/teridentifikasi. Untuk setiap game_map tersebut ditentukan arah tujuannya berdasarkan koordinat simpul yang diinputkan saat ini. Apabila masih belum mencapai control room yang mana maka lakukan BFS dengan simpul awalnya koordinat simpul saat ini dan goal/tujuannya area yang belum teridentifikasi, bila hasil dari BFS adalah None berarti seluruh area pada game_map tersebut telah teridentifikasi sehingga lakukan BFS dari koordinat simpul saat ini menuju ke control room. Bila telah mencapai control room maka lakukan BFS dari control room menuju ke posisi awal.

C. *Implementasi Algoritma BFS dalam menyelesaikan Permainan The Labyrinth dan Pengujian*

Dalam mengimplementasikan algoritma BFS dibuat fungsi-fungsi tambahan untuk membantu dalam penyelesaian masalah.

1. Fungsi is_on_map(v), yaitu fungsi boolean yang mengecek apakah koordinat simpul v berada pada map

```
# mengecek apakah koordinat simpul v berada pada bidang map
def is_on_map(v):
    if v[0] < 0 or v[0] >= r or v[1] < 0 or v[1] >= c:
        return False
    return True
```

labirin atau tidak.

Gambar 11. Implementasi Fungsi is_on_map

(Sumber : Hubbert Jarrosz, 2016)

2. Fungsi get_neighbours(vertex), yaitu fungsi yang mengembalikan tetangga-tetangga dari simpul vertex

```
# mengembalikan map field/ tetangga-tetangga
# dari simpul vertex yang berada pada bidang map
def get_neighbours(vertex):
    vr, vc = vertex
    first_set = {(vr-1, vc), (vr+1, vc), (vr, vc-1), (vr, vc+1)}

    return {v for v in first_set if is_on_map(v)}
```

yang masih berada pada map.

Gambar 12. Implementasi Fungsi get_neighbours(vertex)

(Sumber : Hubbert Jarrosz, 2016)

3. Fungsi first_step(parent, start, n), yaitu fungsi yang nantinya akan digunakan untuk melacak/mencari *step*/langkah untuk kembali ke posisi awal.

```
# Melacak/Mencari step/langkah untuk kembali ke posisi awal
def first_step(parent, start, n):
    v = n
    while parent[v[0]][v[1]] != start:
        v = parent[v[0]][v[1]]
    return v
```

Gambar 13. Implementasi Fungsi first_step(parent, start, n)

(Sumber : Hubbert Jarrosz, 2016)

4. Fungsi where_to_go(game_map, start, back), yaitu fungsi yang digunakan untuk menentukan arah tujuan Kirk(karakter dalam game ini) akan pergi.

```
# menentukan arah tujuan dari Kirk
def where_to_go(game_map, start, back):
    if not back:
        go_to = BFS(game_map, start, '?')
        if go_to == None:
            return BFS(game_map, start, 'C')
        return go_to
    else:
        return BFS(game_map, start, 'T')
```

Gambar 14.Implementasi Fungsi where_to_go

(Sumber : Hubber Jarrosz, 2016)

Implementasi algoritma BFS adalah sebagai berikut, setelah dilakukan inisialisasi untuk algoritma BFS, traversal dimulai dari simpul v saat ini, kemudian inisialisasi colour pada koordiant simpul saat ini dengan 1 dan array distance nya 0. Kemudian masukkan simpul tersebut kedalam antrian, selama antrian tidak kosong lakukan pengecekan apakah sel map pada koordinat simpul tersebut berisi '?' yang berarti belum diidentifikasi, jika iya maka tetangga-tetangga dari simpul tersebut yang isi sel-nya bukan '#' dan 'C', jika tidak maka tetangga-tetangga dari simpul tersebut yang isi sel-nya bukan '#', kemudian untuk setiap tetangga dari sel tersebut, masukkan kedalam antrian, apabila telah ditemukan control room (isi sel tetangga dari simpul tersebut 'C') maka lakukan traceback/ Kembali ke posisi awal.

Algoritma BFS ini memiliki kompleksitas waktu pada umumnya yaitu O(|V| + |E|). Berikut implementasi dari algoritma BFS

```
# Algoritma BFS untuk The Labyrinth game
def BFS(game_map, start, goal):

    # preparation
    queue = []
    colour = []
    distance = []
    parents = []
    for row in range(len(game_map)):
        colour.append([])
        distance.append([])
        parents.append([])
        for collumn in range(len(game_map[row])):
            colour[row].append(0)
            distance[row].append(INFINITY)
            parents[row].append(None)

    colour[start[0]][start[1]] = 1
    distance[start[0]][start[1]] = 0

    queue.append(start)
```

Gambar 15. Implementasi Algoritma BFS bagian 1

(Sumber : Hubbert Jarrosz, 2016)

```
    # melakukan penelusuran ke tetangga-tetangga yang masuk ke antrian
    while queue != []:
        u = queue.pop(0)
        forbidden_symbols = ['#']
        if goal == '?':
            forbidden_symbols.append('C')
        neighbours = {x for x in get_neighbours(u)
                      if game_map[x[0]][x[1]] not in forbidden_symbols}
        for n in neighbours:
            if colour[n[0]][n[1]] == 0:
                colour[n[0]][n[1]] = 1
                distance[n[0]][n[1]] = distance[u[0]][u[1]] + 1
                parents[n[0]][n[1]] = u
                queue.append(n)
                if game_map[n[0]][n[1]] == goal:
                    return first_step(parents, start, n)
        colour[u[0]][u[1]] = 2

    return None
```

Gambar 16. Implementasi Algoritma BFS bagian 2

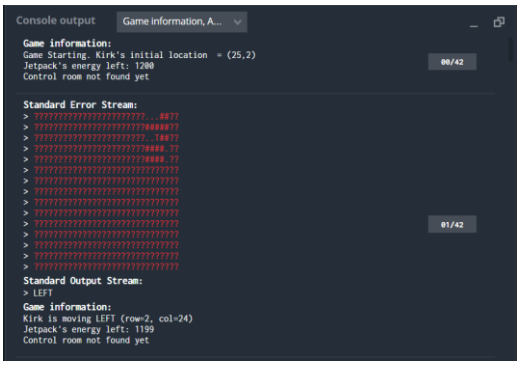
(Sumber : Hubbert Jarrosz, 2016)

Pengujian

'T' adalah posisi awal Kirk, '#'menyatakan tembok, '.' menyatakan ruang hampa, '?' menyatakan area yang belum discan/identifikasi, dan 'C' menyatakan control room.

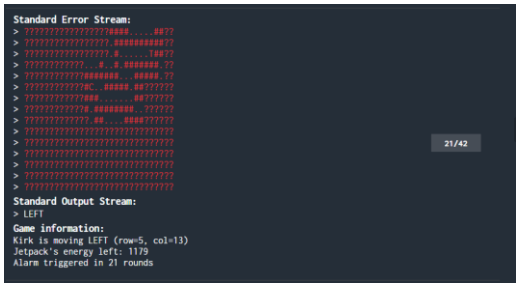
Gambar 17. Peta Labirin Permainan The Labyrinth

(Sumber : Dokumentasi Penulis)



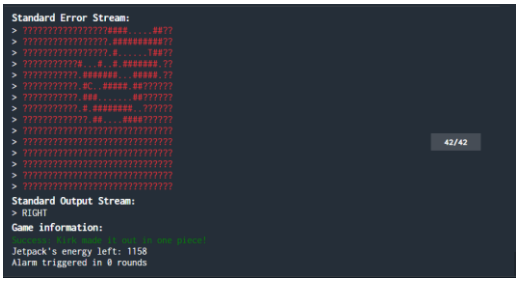
Gambar 18. Struktur Peta Labirin di awal

(Sumber : Dokumentasi Penulis)



Gambar 19. Peta Labirin saat telah ditemukan control room

(Sumber : Dokumnetasi Penulis)



Gambar 20. Peta Labirin saat telah kembali ke posisi awal

(Sumber : Dokumentasi Penulis)

Pada Gambar 18. Struktur Peta Labirin di awal, awalnya akan dia berada pada posisi (25,2), karena belum ditemukan control room, maka arah tujuan penulurannya menuju ke sel '?', sehingga dilakukan BFS dengan simpul awal (25,2) ke '?', kemudian melakukan pendataan tetangga-tetangga dari simpul (25,2) yang tidak berisi '#' dan 'C' karena memang belum ada. Kemudian tetangga-tetangga tersebut dimasukkan kedalam antrian, dan dilakukan penelusuran ke tetangga-tetangganya. Karena belum bertemu control room, maka algoritma BFS ini akan mengembalikan None, kemudian akan melakukan BFS dari simpul saat ini dengan tujuan menuju control room, dan menghasilkan arah LEFT menuju (24,2), hal ini terus berlanjut hingga Pada Gambar 19. Peta Labirin saat telah ditemukan control room, karena telah ditemukan control room pada koordinat (13,5) maka dilakukan BFS dengan simpul awal (13,5) menuju posisi awal (25,2) seperti pada Gambar 20. Peta Labirin saat telah kembali ke posisi awal.

IV. KESIMPULAN

Algoritma BFS merupakan algoritma pencarian yang melakukan traversal pada graf secara menyeluruh tanpa menggunakan informasi tambahan. Algoritma ini bisa digunakan muntuk menyelesaikan permainan The Labyrinth.

VIDEO LINK AT YOUTUBE *(Heading 5)*

https://youtu.be/dtbq3au7SJc

UCAPAN TERIMAKASIH

Puji syukur penulis panjatkan pada hadirat Allah SWT. Atas rahmat dan kemudahan yang diberikan kepada penulis, sehingga makalah yang berjudul "Penerapan Algoritma BFS dalam Penyelesaian Permaian The Labyrinth" dapat diselesaikan. Banyak pihak yang telah membantu dalam penyusunan makalah ini. Oleh karena itu penulis ingin menyampaikan rasa syukur dan terima kasih kepada Dr.Ir. Rila Mandala, M.Eng. selaku Dosen Mata Kuliah IF2211 Strategi Algoritma dan teman-teman seperjuangan yang telah memberikan banyak masukan, referensi, dan dukungan.

DAFTAR PUSTAKA

[1] R.Munir, *Slide Graf bag.1*(2020), Bandung.

[2] R.Munir, *Slide Graf bag.2*(2020), Bandung.

[3] R. Munir, N.U.Maulidevi, *Breadth/Depth First Search (BFS/DFS)bag.1, Bandung.*

[4] https://github.com/vadim-job-hg/Codingame/blob/master/HARD/THE%20LABYRINTH/python3/the-labyrinth.py

Diakses pada tanggal 3 Mei 2020 pukul 19.00 WIB.

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, 5 Mei 2021

Mochammad Fatchur Rochman
13519009