Optimasi Rute Beacon pada Game Faster Than Light dengan Directed Acyclic Graph dan Depth-First Search

Reinhard Mikhael Tandra - 13525076
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: sayareinhard@gmail.com , 13525076@std.stei.itb.ac.id

***Abstract*— Faster Than Light adalah game roguelike yang bertemakan luar angkasa. Pemain dapat mendatangi beacon-beacon untuk mendapatkan scrap yang digunakan untuk mengupgrade kapal pemain. Di sektor terakhir, pemain akan berhadapan dengan rebel flagship yang kuat. Untuk menang melawan rebel flagship, pemain harus dapat memaksimalkan upgrade kapalnya. Selain itu, di tiap sektor sebelum sektor terakhir, fleet rebel akan maju secara perlahan dan menguasai beacon secara perlahan sampai semua beacon di sektor itu dikuasai rebel. Daerah yang sudah dikuasai oleh rebel tidak akan memberikan scrap. Maka dari itu, pemain dapat menerapkan teori Directed Acyclic Graph dan algoritma Depth First Search untuk memaksimalkan beacon yang dilewati (*Abstract*)**

Keywords—Graph; Directed Acyclic Graph; Depth First Search; Beacon

I. PENDAHULUAN

FTL: Faster Than Light adalah game yang dirilis pada tanggal 15 September 2012 dengan genre roguelike dan strategy. Roguelike adalah genre yang berasal dari game Rogue yang rilis di tahun 1980. Game Rogue memiliki sifat yang unik dimana konten didalam game tersebut di generasi secara prosedural dan pasti akan berbeda setiap kali dimainkan.

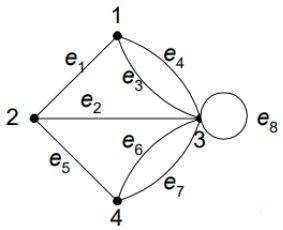
Faster Than Light juga menggunakan generasi prosedural dalam pembuatan tiap sektornya. Tiap sektor akan memiliki 19-24 beacon yang saling sambung-menyambung antara satu sama lain. Tiap sektor hanya akan memiliki satu beacon exit untuk menuju ke sektor berikutnya. Apabila pemain terlalu lama didalam satu sektor, rebel fleet akan menguasai semua beacon di sektor tersebut dan pemain akan harus melawan musuh yang sulit tanpa mendapatkan hadiah yang setimpal.

Hubungan antara beacon pada tiap sektor dapat di visualisasikan menjadi suatu konsep pada Matematika Diskrit yaitu graf. Lebih tepatnya, Directed Acyclic Graph untuk memaksimalkan beacon yang dilewati tanpa kembali ke beacon yang sudah dilewati. Apabila pemain sudah mencapai beacon exit, pemain harus segera meninggalkan sektor tersebut untuk meminimalisir penggunaan bahan bakar. Selain itu, hal yang perlu diperhatikan adalah rebel fleet yang akan maju satu langkah dan menguasai beberapa beacon setiap kali pemain berpindah beacon. Makalah ini akan menggunakan algoritma Depth First Search untuk memaksimalkan beacon yang dilewati tiap sektornya.

II. LANDASAN TEORI

A. Graf

Pada umumnya, graf digunakan untuk merepresentasikan objek-objek dan hubungannya antara satu sama lain. Graf terdiri dari simpul (vertex) dan sisi (edge). Apabila simpul adalah daratan, maka sisi adalah jembatan-jembatan yang saling menghubungi daratan. Graf G dapat didefinisikan sebagai G = (V, E), dimana V adalah simpul dan E adalah sisi.



G_3

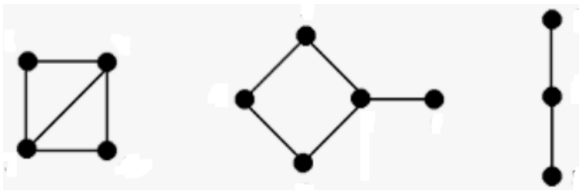
Gambar 1. Graf G_3, diambil dari [3].

Pada graf G_3, sisi $e_3 = (1,3)$ dan sisi $e_4 = (1,3)$ dinamakan sisi ganda karena menghubungkan dua simpul yang sama. Sedangkan, sisi $e_8 = (3,3)$ dinamakan sisi gelang (loop) karena berawal dan berakhir di simpul yang sama

Graf memiliki beberapa jenis berdasarkan sifatnya, diantaranya ada :

1) **Graf Sederhana**

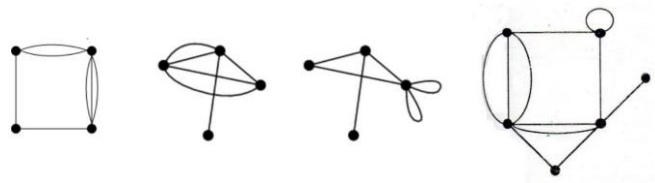
Graf yang tidak memiliki sisi gelang maupun sisi ganda.



Gambar 2. Graf Sederhana diambil dari [3].

2) **Graf Tak Sederhana**

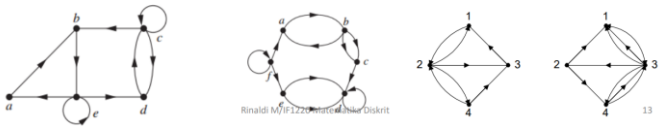
Graf yang mengandung sisi gelang atau sisi ganda



Gambar 3. Graf Tak Sederhana diambil dari [3].

3) **Graf Berarah**

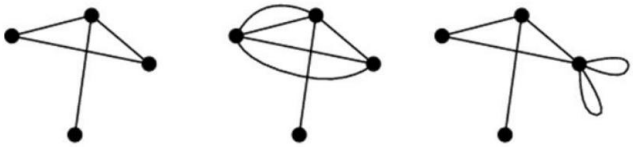
Graf yang tiap sisinya memiliki arah keluar dari simpul dan masuk ke simpul



Gambar 4. Graf Berarah diambil dari [3].

4) **Graf Tak Berarah**

Graf yang sisinya tidak memiliki arah masuk atau keluar



Gambar 5. Graf Tak Berarah diambil dari [3].

Graf juga memiliki beberapa istilah atau terminologi yang dapat digunakan seperti :

- Ketetanggaan (Adjacent)
  Dua buah simpul dikatakan bertetangga ketika ada satu sisi yang menghubungkan kedua simpul

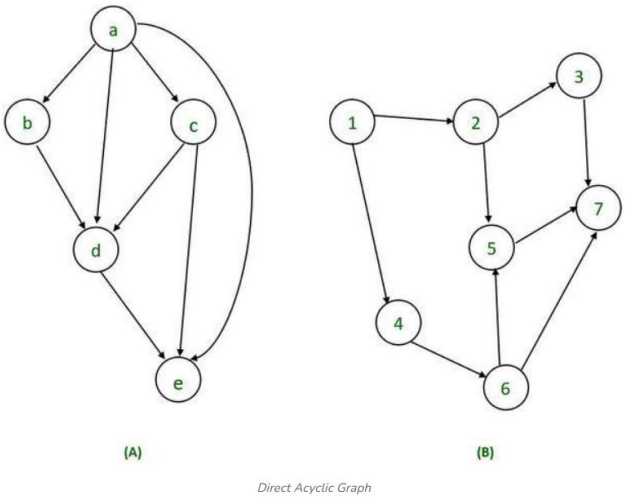
- Bersisian (Incidency)
  Sisi e bersisian dengan kedua simpul yang dihubungkan

- Simpul Terpencil (Isolated Vertex)
  Simpul yang tidak bersisian dengan sisi apapun

- Graf Kosong (Null Graph/Empty Graph)
  Graf yang tidak memiliki sisi

- Derajat (Degree)
  Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut

B. Directed Acyclic Graph (DAG)

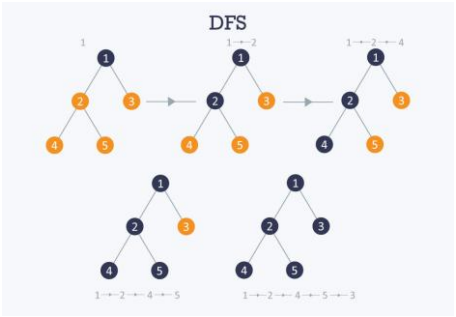


Gambar 6. Directed Acyclic Graph diambil dari [4].

Directed Acyclic Graph atau graf asiklik berarah adalah salah satu contoh dari graf berarah. Namun, graf asiklik berarah memiliki sifat yang berbeda dengan graf berarah pada umumnya. Graf berarah memungkinkan grafnya untuk memiliki siklus, namun graf asiklik berarah tidak memperbolehkan graf untuk memiliki siklus. Setiap graf asiklik berarah pasti memiliki satu titik awal dan satu titik akhir.

C. Depth-First Search (DFS)

Depth First Search atau DFS adalah algoritma yang dapat digunakan untuk menyelesaikan masalah ini. Cara kerja Depth First Search adalah dengan menelusuri setiap simpul secara satu per satu dan dapat melakukan backtrack ke simpul sebelumnya untuk mencari simpul lain yang belum dilalui.



Gambar 7. Depth First Search, diambil dari [6]

Pada umumnya, Depth First Search adalah algoritma yang menggunakan rekursi untuk mendapatkan hasil akhirnya. Simpul-simpul yang sudah dilalui akan diberikan suatu tanda. Tanda pada simpul berfungsi supaya semua simpul hanya dilalui sebanyak satu kali saja supaya tidak menimbulkan loop tanpa batas.

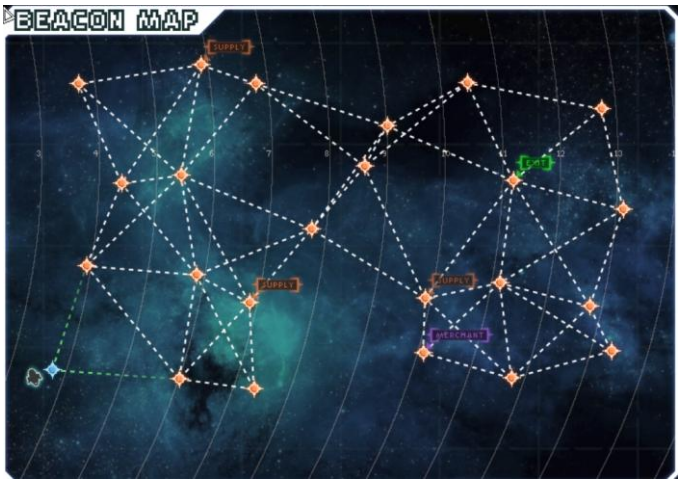
D. Memoization

Memoization atau memoisasi adalah suatu teknik yang dapat digunakan untuk mengoptimasi performa algoritma. Memoisasi umumnya digunakan untuk membuat algoritma yang bersifat rekursif menjadi lebih efisien. Teknik ini dilakukan dengan menyimpan hasil dari pemanggilan suatu fungsi rekursif untuk bisa digunakan kembali saat dipanggil lagi. Teknik Memoisasi akan mempercepat program rekursif karena program tidak perlu memanggil fungsi yang sama secara berulang kali.

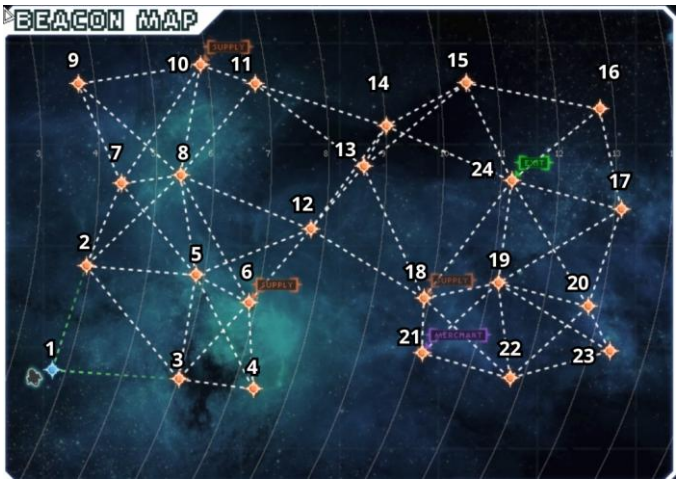
III. PEMBAHASAN

A. Membentuk Directed Acyclic Graph dari Peta Sektor pada Faster Than Light

Beacon yang berada pada peta sektor adalah simpul dan jalur antara beacon adalah sisi berarah. Directed Acyclic Graph dimulai dari beacon pertama dan berakhir di beacon exit. Peta sektor dibuat menjadi Directed Acyclic Graph karena pemain ingin melalui beacon sebanyak mungkin sebelum rebel fleet menguasai beacon exit.

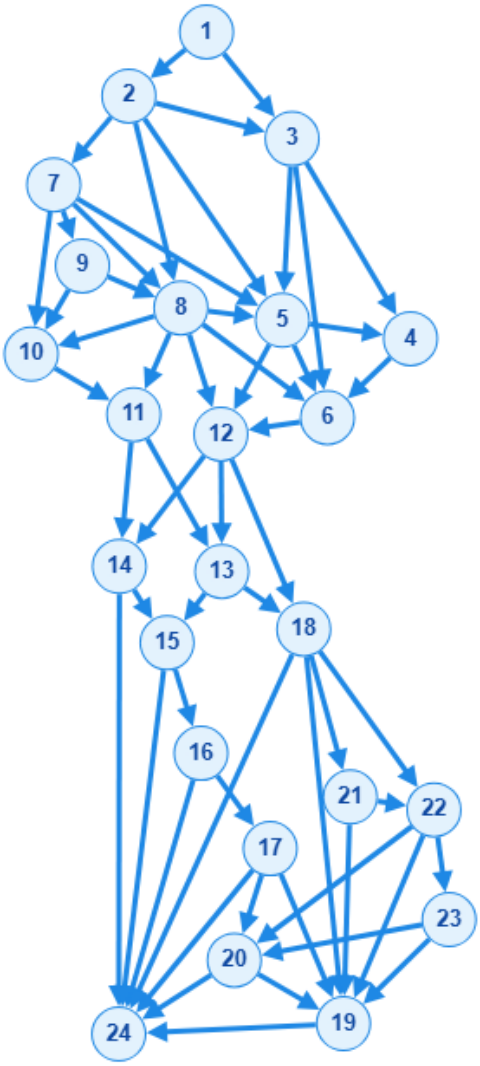


Gambar 8. Map Sektor pada Faster Than Light



Gambar 9. Map Sektor berindeks pada Faster Than Light

Beacon dengan nomor 1 adalah beacon start dan beacon dengan nomor 24 adalah beacon exit yang kita tuju. Apabila background dari peta diperhatikan, terdapat berbagai setengah lingkaran dan angka-angka yang memiliki fungsi untuk memberikan informasi kepada pemain tentang kapan rebel fleet akan menguasai beacon-beacon yang berada di belakang setengah lingkaran tersebut. Melalui informasi ketetanggaan beacon, dapat dibuat Directed Acyclic Graph dan dari setengah lingkaran di background peta, dapat ditarik informasi tentang kapan rebel menguasai beacon tersebut.



Gambar 10. Representasi Peta Sektor pada Faster Than Light sebagai Directed Acyclic Graph, dibuat dengan https://graphonline.top/.

B. *Struktur Data*

```
#define MAX_NODE 25

int adjacency_matrix[MAX_NODE][MAX_NODE];
int memo[MAX_NODE][MAX_NODE];
typedef struct{
    int rebel[MAX_NODE];
    int nEff;
} node;
```

Gambar 11. Struktur Data

Pada satu sektor, jumlah node yang maksimal adalah 24 node sehingga MAX_NODE di define sebagai 25.

adjacency_matrix[MAX_NODE][MAX_NODE] adalah representasi dari hubungan ketetanggaan antar simpul pada Directed Acyclic Graph dengan adjacency_matrix[i][j] dengan i adalah indeks beacon awal dan j adalah indeks beacon akhir.

memo[MAX_NODE][MAX_NODE] adalah implementasi dari teknik memoisasi yang mencatat berapa beacon maksimal yang dapat dilalui dari indeks dan turn tersebut sebelum sampai ke node exit dan juga sebelum rebel fleet tiba dengan memo[i][j] dengan i adalah indeks beacon dan j adalah turn.

typedef struct node digunakan untuk menyimpan berapa jumlah node yang ada pada map sektor di node.nEff dan untuk menyimpan informasi tentang kapan rebel akan tiba di setiap beacon di node.rebel[i] dengan i adalah indeks beacon.

```
int exit_node;
node NODE;
```

Gambar 12. Inisialisasi variabel

Penginisialisasian variabel dilakukan di luar int main dan fungsi supaya variabel global dapat digunakan didalam fungsi dan juga int main.

C. *Inisialisasi*

```
NODE.nEff = 24;
```

Gambar 13. Kode jumlah beacon

Menyimpan jumlah beacon yang ada pada peta sektor.

```
NODE.rebel[1] = 4; NODE.rebel[2] = 4; NODE.rebel[9] = 4;
NODE.rebel[7] = 5;
NODE.rebel[3] = 6; NODE.rebel[5] = 6; NODE.rebel[8] = 6; NODE.rebel[10] = 6;
NODE.rebel[6] = 7; NODE.rebel[11] = 7;
NODE.rebel[4] = 8; NODE.rebel[12] = 8;
NODE.rebel[13] = 9;
NODE.rebel[14] = 10; NODE.rebel[18] = 10;
NODE.rebel[21] = 11; NODE.rebel[15] = 11;
NODE.rebel[22] = 12; NODE.rebel[19] = 12; NODE.rebel[24] = 12;
NODE.rebel[20] = 13; NODE.rebel[16] = 13;
NODE.rebel[23] = 14; NODE.rebel[17] = 14;
```

Gambar 14. Kode kedatangan rebel pada tiap beacon

Menyimpan informasi tentang turn fleet rebel akan menguasai beacon setiap beacon pada peta.

Gambar 15. Kode ketetanggan beacon

Menyimpan hubungan ketetanggaan antar beacon yang bersifat Direct Acyclic Graph.

```
int i, j;
for(i = 0; i < MAX_NODE; i++){
    for (j = 0; j < MAX_NODE; j++){
        memo[i][j] = -1;
        adjacency_matrix[i][j] = 0;
    }
}
```

Gambar 16. Kode inisialisasi memo dan adjacency matrix

Penginisialisasian isi memo dengan -1 dan adjacency matrix dengan 0

```
exit_node = 24;
```

Gambar 17. Kode indeks exit node

Menginisialisasi indeks beacon yang berlaku sebagai exit node agar algoritma tahu kapan harus berhenti

```
int ans;
ans = dfs(1, 0);
printf("Jumlah max beacon yang dapat dilalui : %d", ans);
```

Gambar 18. Kode output DFS

Penginialisasian int ans dan pemanggilan dfs(1, 0) yaitu dari beacon dengan indeks 1 pada turn 0 untuk memulai proses

Depth-First Search untuk mencari jumlah maksimal beacon yang dapat dilalui. Setelah proses Depth-First Search selesai, hasilnya akan ditampilkan ke layar.

D. Pengimplementasian Algoritma Depth-First Search dan Memoisasi

```
int dfs(int idx, int turn){
```

Gambar 19. Fungsi DFS

Fungsi dfs(int idx, int turn) adalah fungsi utama yang akan digunakan dan juga bersifat rekursif. Fungsi ini menggunakan parameter idx dan turn dimana idx adalah beacon ke-idx yang sedang dicari dan turn adalah turn saat algoritma sedang berjalan.

```
if(NODE.rebel[idx] <= turn){
    return -1;
}
```

Gambar 20. Kode if rebel

If block ini terjadi saat rebel telah menguasai atau tiba di beacon yang sama dengan pemain dan akan langsung mengembalikan nilai -1 untuk menandakan bahwa kombinasi antara indeks dan turn ini akan menyebabkan pemain masuk ke beacon yang dikuasai oleh rebel fleet.

```
if(idx == exit_node){
    return 0;
}
```

Gambar 21. Kode if exit

If block ini terjadi saat pemain sampai ke exit node dan langsung mengembalikan 0.

```
if(memo[idx][turn] != -1){
    return memo[idx][turn];
}
```

Gambar 22. Kode if memo

If block ini terjadi saat pemanggilan fungsi dengan idx dan turn tertentu sudah dilakukan sebelumnya sehingga langsung mengambil dan mengembalikan hasilnya dari memo. Teknik ini adalah teknik memoisasi yang dilakukan untuk mengefisiensikan fungsi rekursif seperti fungsi ini.

```
int i, temp = -1, max = -1;
for(i = 1; i <= NODE.nEff; i++){
    if(adjacency_matrix[idx][i] == 1){
        temp = dfs(i, turn + 1);
        if(temp > max){
            max = temp;
        }
    }
}
```

Gambar 23. Kode for loop untuk DFS

Variabel lokal int i, int temp, dan int max diinisialisasi dan variabel temp dan max diisi angka -1. For loop akan berjalan dari indeks 1 sampai dengan indeks 24. Node.nEff adalah jumlah dari beacon pada sektor tersebut. Semua simpul yang bertetangga dengan simpul idx akan dipanggil fungsinya dengan paramater idx sebagai simpul yang bertetanggaan dan parameter turn yang ditambah 1. Hal ini seolah-olah pemain mengambil semua kemungkinan rute beacon dapat mencapai exit. Hasil pemanggilan terbesar akan dimasukkan ke dalam variabel max.

```
if (max == -1){
    memo[idx][turn] = -1;
    return -1;
}else{
    memo[idx][turn] = max + 1;
    return max + 1;
}
```

Gambar 24. Kode memoisasi

Apabila algoritma tidak dapat menemukan tetangga dari simpul yang belum dikuasai oleh rebel, maka max akan tetap bernilai -1 sehingga masuk ke if block pertama dan memo juga pengembalian fungsi pada idx dan turn tersebut adalah -1.

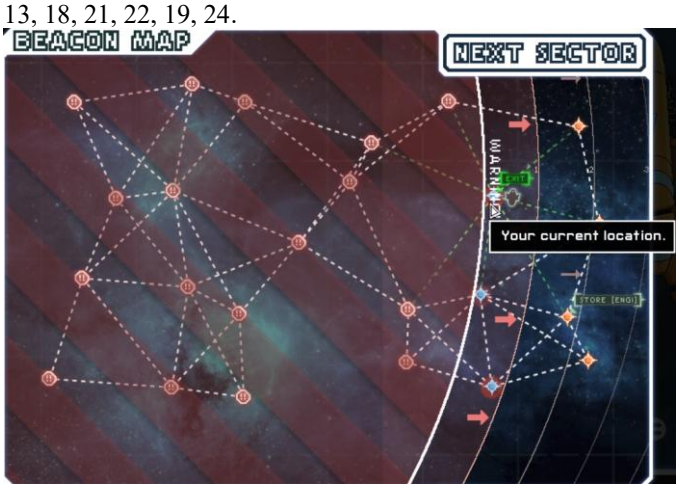
Sebaliknya, apabila hasil dari pemanggilan rekursif mencapai exit, maka max tidak akan -1 sehingga hasil dari proses rekursif tadi akan ditambah 1 dan dikembalikan juga disimpan ke memo. Hasil max ditambah dengan 1 karena kapal berhasil berpindah beacon dan proses rekursif bisa mengakumulasi nilainya.

IV. KESIMPULAN

```
Jumlah max beacon yang dapat dilalui : 11
```

Gambar 25. Hasil run Kode

Menurut hasil run kode diatas, maksimal beacon yang dapat dilalui oleh pemain adalah 11 beacon. Maka dari itu, pemain dapat pergi ke 11 beacon berbeda tanpa masuk ke beacon yang sudah dikuasai oleh rebel fleet. Salah satu rute yang dapat diambil oleh pemain dengan memaksimalkan beacon yang dilalui dengan dimulai dari 1 adalah 2, 3, 4, 6, 12, 13, 18, 21, 22, 19, 24.



Gambar 26. Hasil Mengikuti Rute dari DFS

Dengan mengikuti rute tersebut, pemain dapat melewati 11 beacon dengan sampai ke beacon exit satu turn sebelum rebel menguasai beacon exit. Melalui algoritma Depth-First Search, pemain dapat melalui beacon sebanyak-banyaknya dengan aman dan mengupgrade kapalnya secara maksimal untuk memenangkan permainan

REFERENCES

[1] Subset Games, "FTL: Faster Than Light", https://store.steampowered.com/app/212680/FTL_Faster_Than_Light/, diakses pada 11 Juni 2026

[2] Tim Brookes, "Roguelikes: A Unique & Challenging Spin On The RPG Genre", https://www.makeuseof.com/tag/roguelikes-a-unique-challenging-spin-on-the-rpg-genre/, diakses pada 11 Juni 2026.

[3] R. Munir, "Graf (Bagian 1)", https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf, diakses pada 11 Juni 2026.

[4] GeeksForGeeks, "Introduction to Directed Acyclic Graph", https://www.geeksforgeeks.org/dsa/introduction-to-directed-acyclic-graph/, diakses pada 13 Juni 2026.

[5] GeeksForGeeks, "Depth First Search or DFS for a Graph", https://www.geeksforgeeks.org/dsa/depth-first-search-or-dfs-for-a-graph/, diakses pada 13 Juni 2026

[6] HackerEarth, "Depth First Search", https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/, dikases pada 14 Juni 2026

[7] GeeksForGeeks, "What is memoization? A Complete Tutorial", https://www.geeksforgeeks.org/dsa/what-is-memoization-a-complete-tutorial/, diakses pada 14 Juni 2026

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, 19 Juni 2026

Reinhard Mikhael Tandra, 13525076