LAPORAN TUGAS BESAR AKA
BREADTH-FIRST SEARCH vs. DEPTH-FIRST SEARCH (ITERATIVE vs. RECURSIVE)

Daniyal Arshaq Sudrajat	103022330076
Riziq Rizwan	103022300119

S1 REKAYASA PERANGKAT LUNAK
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY BANDUNG
2024

DAFTAR ISI

BAB I

1. Pendahuluan

Catur merupakan permainan strategi antar dua pemain di papan kotak-kotak hitam dan putih yang disusun dalam ukuran 8×8, atau 64 kotak total. Permainan ini dikelompokkan menjadi dua pemain; kelompok hitam dan kelompok putih. Catur sudah dimainkan oleh jutaan orang di seluruh dunia dan telah menjadi permainan yang sinonim dengan kepintaran, kelicikan, dan kemampuan mental. Pada mula permainan, setiap pemain memiliki 16 buah catur: satu raja, satu ratu, dua benteng, dua kuda, dua bishop, dan delapan bidak. Setiap buah catur memiliki gaya gerakannya masing-masing, dengan ratu yang paling kuat dan bidak paling lemah.

Kuda merupakan salah satu buah catur paling unik, karena gerakannya yang membentuk huruf "L". Studi kasus ini akan berfokus pada menghitung jumlah cara kuda dapat mencapai suatu kotak tertentu pada papan catur berukuran N×N posisi awal tertentu (X dan Y), dalam beberapa langkah tertentu dan dihitung dalam milidetik. Studi kasus ini bisa digunakan untuk mempelajari pola gerakan kuda serta penerapan algoritma pencarian jalur dalam lingkungan *Java* dan *integrated development environment* (IDE) NetBeans oleh *Apache Software Foundation*. Perlu dicatat bahwa studi kasus ini hanya mengukur waktu yang dibutuhkan untuk eksplorasi pola gerakan kuda pada sebuah ukuran papan, koordinat awal dan koordinat tujuan kuda di papan yang telah ditentukan di dataset. Program hanya menerima input size langkah yang dimasukkan *user* untuk memproses data.

2. Definisi dan Pseudocode

Pada studi kasus ini, akan memfokuskan pada dua algoritma traversal, yaitu algoritma pencarian iteratif Breadth-First Search dan rekursif Depth-First Search.

2.1. Breadth-First Search (BFS)

Breadth-First Search adalah algoritma iteratif traversal atau pencarian yang seringkali digunakan untuk menjelajahi struktur data seperti Tree, Graph, Array 2D, dan Array 3D. BFS bekerja dengan mengeksplorasi semua simpul pada level yang sama sebelum melanjutkan

pencarian ke level berikutnya. BFS seringkali digunakan untuk menemukan jalur terpendek pada sebuah grid karena sifat eksplorasi levelnya. BFS menjamin akan menemukan jalur terpendek, tetapi dengan pengorbanan penggunaan lebih banyak memori karena menggunakan queue.

2.2. Depth-First Search (DFS)

Depth-First Search adalah algoritma rekursif traversal atau pencarian yang seringkali digunakan untuk menjelajahi struktur data seperti Tree, Graph, Array 2D, dan Array 3D. DFS bekerja dengan mengeksplorasi jalur sedalam mungkin terlebih dahulu sebelum kembali untuk memproses jalur lain. DFS tidak bisa menjaminkan menemukan jalur terpendek, tetapi lebih hemat memori dibanding dengan Breadth-First Search.

Perbedaan BFS dan DFS		
Kriteria	**Breadth-First Search (BFS)**	**Depth-First Search (DFS)**
Struktur	Iteratif	Rekursif
Traversal	Level-order	Depth-order
Kompleksitas Waktu	$O(\|V\| + \|E\|)$	$O(\|V\| + \|E\|)$
Aplikasi	Jalur terpendek dan pencarian level	Eksplorasi Lengkap

Tabel 2.2.1.

2.3. Pseudocode Breadth-First Search (Iteratif)

```java
public class GerakanKuda {
    static int[] dx = {-2, -2, -1, -1, 1, 1, 2, 2};
    static int[] dy = {-1, 1, -2, 2, -2, 2, -1, 1};

    public int HitungLangkahBFS_Iteratif(int ukuranPapan, int awalX, int awalY, int tujuanX, int tujuanY, int jumlahLangkah) {
        /*
        n = Ukuran papan catur (N x N)
        (x1, y1) = Koordinat awal kuda
        (x2, y2) = Koordinat tujuan kuda
        k = jumlah step yang diinginkan
        */

        // representasi array 3D
        int[][][] dp = new int[ukuranPapan][ukuranPapan][jumlahLangkah + 1];
        dp[awalX][awalY][0] = 1;

        for (int langkahSaatIni = 1; langkahSaatIni <= jumlahLangkah; langkahSaatIni++) {
            for (int posisiX = 0; posisiX < ukuranPapan; posisiX++) {
                for (int posisiY = 0; posisiY < ukuranPapan; posisiY++) {
                    if (dp[posisiX][posisiY][langkahSaatIni - 1] > 0) {
                        for (int arah = 0; arah < 8; arah++) {
                            int posisiBaruX = posisiX + dx[arah];
                            int posisiBaruY = posisiY + dy[arah];
                            if (posisiBaruX >= 0 && posisiBaruX < ukuranPapan && posisiBaruY >= 0 && posisiBaruY < ukuranPapan) {
                                dp[posisiBaruX][posisiBaruY][langkahSaatIni] += dp[posisiX][posisiY][langkahSaatIni - 1];
                            }
                        }
                    }
                }
            }
        }

        int totalLangkah = 0;
        for (int langkahSaatIni = 0; langkahSaatIni <= jumlahLangkah; langkahSaatIni++) {
            totalLangkah += dp[tujuanX][tujuanY][langkahSaatIni];
        }
        return totalLangkah;
    }
```

Gambar 2.3.1.

2.3.1. Tujuan

Breadth-First Search pada program bertujuan untuk menghitung jumlah cara untuk memindahkan kuda dari koordinat awal (awalX, awalY) ke posisi koordinat tujuan (tujuanX, tujuanY) dalam jumlah langkah tertentu (k) pada papan catur berukuran N×N.

2.3.2. Pendekatan

Breadth-First Search menggunakan metode iteratif dengan matriks 3D yang merepresentasikan:

- Dimensi 1 (x): Koordinat X pada papan.
- Dimensi 1 (y): Koordinat Y pada papan.
- Dimensi 1 (k): Langkah.

2.3.3. Langkah-Langkah

1. Inisialisasi Matriks dp:
    - Nilai diinisialisasi menjadi 0.
    - $dp[awalX][awalY][0] = 1$, karena terdapat 1 cara untuk berada di koordinat awal pada langkah ke-0.
2. Iterasi untuk setiap langkah (k):
    - Untuk setiap langkah, lakukan:
        - Periksa setiap posisi papan ($posisiX, posisiY$) yang valid.
        - Jika posisi tersebut memiliki cara untuk dicapai pada langkah sebelumnya:
            - Periksa semua 8 kemungkinan gerakan kuda di papan.
            - Jika gerakan valid (masih di dalam papan), tambahkan nilai cara dari posisi sebelumnya ke posisi baru.
3. Hitung total cara untuk koordinat tujuan:
    - Jumlahkan semua nilai $dp[tujuanX][tujuanY][langkahSaatIni]$ untuk mendapatkan total cara mencapai posisi tujuan pada langkah k.

2.4. Pseudocode Depth-First Search (Rekursif)

```
public int HitungLangkahDFS_Rekursif(int n, int x, int y, int x2, int y2, int k) {
    // jika langkah abis dan posisi sama dengan tujuan
    if (k == 0) {
        return (x == x2 && y == y2) ? 1 : 0;
    }
    // jika langkah habis tapi posisi belum di tujuan
    if (k < 0) {
        return 0;
    }

    int totalCara = 0;
    // cek 8 kemungkinan gerakan kuda
    for (int i = 0; i < 8; i++) {
        int nextX = x + dx[i];
        int nextY = y + dy[i];
        // memastikan gerakn tetap berada di papan catur
        if (nextX >= 0 && nextX < n && nextY >= 0 && nextY < n) {
            totalCara += HitungLangkahDFS_Rekursif(n, nextX, nextY, x2, y2, k - 1);
        }
    }
    return totalCara;
}
```

Gambar 2.3.2.

2.4.1 Perhitungan

Pencarian kompleksitas waktu pada algoritma di atas dilakukan dengan 2 metode, yaitu:

a. Metode Direct Observation

Tingkat 1 = 8^0 = 1 iterasi/rekursi

2 = 8^1 iterasi/rekursi

3 = 8^2 iterasi/rekursi

Ke i-1 = 8^{i-1}

$T(n,\ k) = 1 + 8^1 + 8^2 + 8^3 + \ldots + 8^n = \frac{(8^{(n+1)}-1)}{(8-1)} \in O(8^n)$.

b. Metode Sum Notation

$T(n)$ = ∑(i=0)^(8^n)8^n -> T(n) = (8^(n+1)-1)/(8-1)

$T(n) \in O(8^n)$.

BAB II

3. Basic Operation

Pada algoritma Breadth-First Search, operasi dasarnya adalah *dp[posisiBaruX][posisiBaruY][langkahSaatIni] += dp[posisiX][posisiY][langkahSaatIni - 1]*. Operasi dasar ini melakukan pemeriksaan bahwa koordinat X dan Y yang baru masih di dalam ruang papan.

Pada algoritma Depth-First Search, operasi dasarnya adalah if (nextX >= 0 && nextX < n && nextY >= 0 && nextY < n) untuk memeriksa apakah posisi baru yang dihitung berada dalam batas papan catur dan apakah langkah yang tersisa masih valid, Operasi ini memastikan bahwa posisi baru (nextX, nextY) yang dihitung setelah pergerakan kuda tetap berada dalam batas papan catur berukuran N×N.

4. Perhitungan Kompleksitas Waktu

Kompleksitas Waktu merupakan ukuran teoritis untuk menggambarkan seberapa banyak waktu yang dibutuhkan sebuah algoritma untuk mengeksekusi sebuah *task* yang berhubungan dengan input *n*. Kompleksitas waktu biasanya diekspresikan dengan notasi asimtotik seperti Big-O ($O(n)$), Big-Omega ($\Omega(n)$), dan Big-Theta ($\Theta(n)$). Kompleksitas Waktu bertujuan untuk menghitung efisiensi algoritma dari segi waktu, serta membantu memahami performa algoritma dalam skenario *worst case*, *best case*, dan *average case*.

4.1. Breadth-First Search

Pada Breadth-First Search, diketahui bahwa setiap iterasi langkah k, algoritma memeriksa setiap posisi papan, memperbarui kemungkinan langkah berdasarkan 8 kemungkinan arah gerakan kuda. Menggunakan rumus time complexity, maka:

$$T_{BFS} = Jumlah\ Langkah \times (Posisi\ pada\ papan) \times (Arah\ Gerakan)$$

Dimana:

- Jumlah Langkah = k.
- Posisi pada papan = $N \times N = N^2$.
- Arah Gerakan = 8.

Sehingga:

$T_{BFS} = k * N^2 * 8$ atau $O(k * N^2)$.

Oleh karena itu, kompleksitas waktu dari Breadth-First Search adalah $O(k * N^2)$.

Perlu ditekankan bahwa kompleksitas waktu ini merupakan hasil dari representasi *matriks array 3D* untuk melacak jumlah cara mencapai posisi tertentu dalam sebuah langkah input size. Dalam iterasi k, setiap posisi (x, y) pada papan N×N dirproses, dan pemeriksaan gerakan dilakukan hingga 8 arah. Oleh karena, itu, kompleksitas waktu yang tepat untuk algoritma ini adalah $O(k * N^2)$. Selain itu, matriks tidak program ini tidak secara eksplisit merepresentasikan simpul V dan sisi E, seperti pada graph tradisional. Kompleksitas ini mencerminkan proses algoritma yang bekerja dengan matriks dan iterasi berdasarkan jumlah langkah, sehingga lebih akurat daripada kompleksitas waktu $O(V + E)$.

4.1. Depth-First Search

Pada Depth-First Search, worst case ketika algoritma harus mengeksplorasi semua jalur dan simpul sebelum menemukan solusi, atau ketika tidak ada solusi sama sekali. Menggunakan rumus time complexity, maka:

$T_{BFS} = (Jumlah\ Simpul) \times (Jumlah\ Sisi)$

$T_{DFS} = N^2 * 8N^2 = O(N^4)$

Jadi, kompleksitas waktu untuk DFS pada papan catur berukuran N×N adalah $O(N^4)$.

5. Notasi Asimtotik

Notasi Asimtotik didefinisikan sebagai suatu notasi yang menggambarkan kelas - kelas pertumbuhan waktu akibat pengaruh masukan data (n) dari algoritma ketika dieksekusi. Notasi Asimtotik sendiri dapat dibagi menjadi 3 jenis, yaitu: Big-O ($O(n)$), Big-Omega ($\Omega(n)$), dan Big-Theta ($\Theta(n)$).

5.1. Breadth-First Search

Karena faktor konstanta 8 diabaikan, maka notasi asimtotik dari algoritma Breadth-First Search adalah $O(k * N^2)$.

5.2. Depth-First Search

Algoritma ini memiliki notasi asimtotik $O(8^n)$,karena pada setiap langkah rekursif, kuda dapat bergerak ke salah satu dari 8 arah yang berbeda.

6. Kelas Efisiensi

Kelas efisiensi sebuah algoritma dapat ditentukan oleh beberapa faktor, antara lain adalah kompleksitas waktu:

6.1. Breadth-First Search

Karena faktor konstanta 8 diabaikan, maka notasi asimtotik dari algoritma Breadth-First Search adalah $O(k * N^2)$.

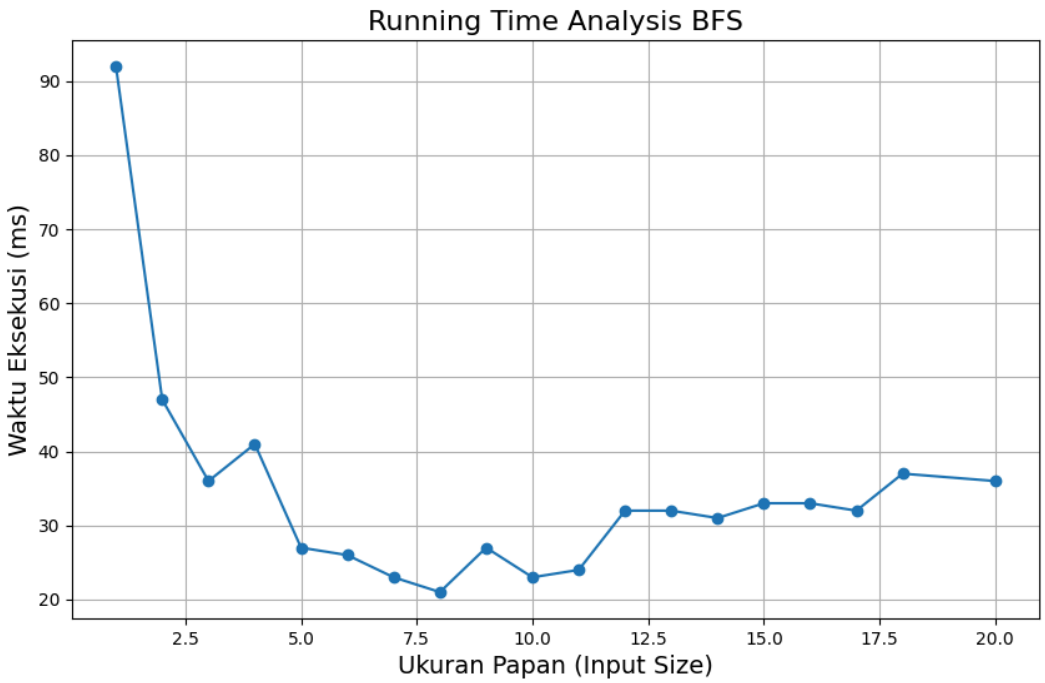
6.2. Depth-First Search

Algoritma ini memiliki kompleksitas waktu $O(8^n)$, di mana k adalah jumlah langkah yang tersisa untuk mencapai tujuan.

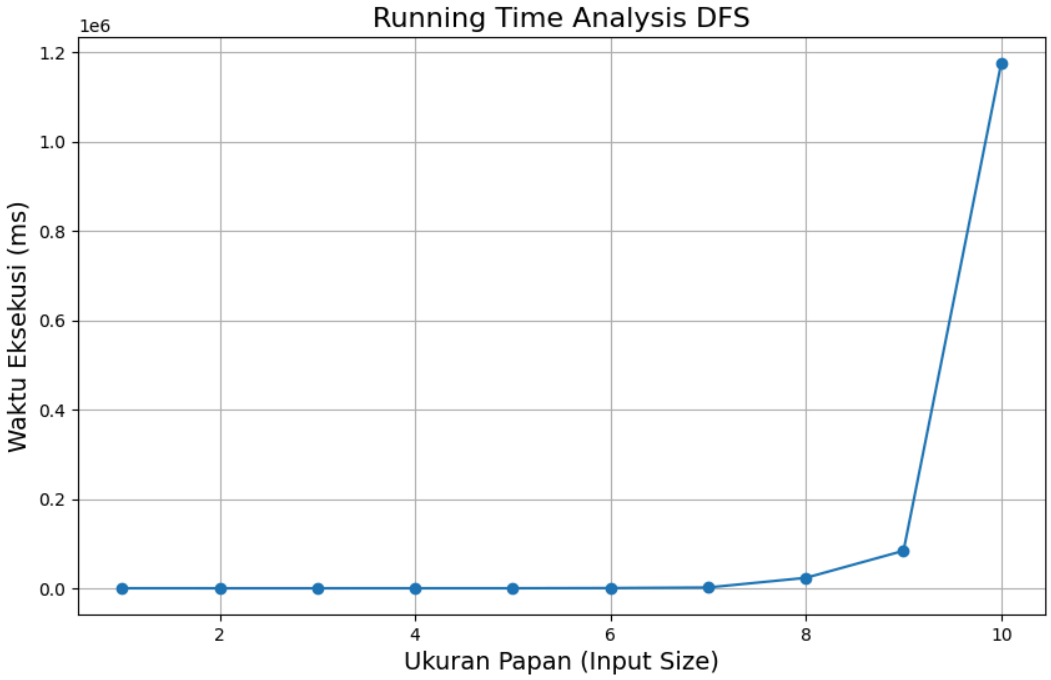
7. Visualisasi Tabel dan Grafik Running Time

Input Size (n)	1	2	3	4	5	6	7	8	9	10
Breadth-First Search	92ms	47ms	36ms	41ms	27ms	26ms	23ms	21ms	27ms	23ms
Depth-First Search	71ms	20ms	19ms	46ms	65ms	478ms	1739 ms	23364 ms	83915 ms	11756 44ms

Tabel 7.1.1.



Gambar 7.1.1.



Gambar 7.1.2.

Grafik line chart dan table running time diatas menunjukkan perbandingan waktu running time antara algoritma Breadth-First Search dan Depth-First Search untuk berbagai ukuran input size (jumlah langkah). Breadth-First Search cenderung memiliki pola waktu eksekusi yang lebih stabil dan konstan dibanding Depth-First Search, yang memiliki variasi jauh lebih besar.

Visualisasi ini juga memunculkan beberapa pertanyaan menarik tentang pendekatan, stabilitas, dan efisiensi dari masing-masing algoritma, seperti:

1. Apa perbedaan pendekatan Breadth-First Search dan Depth-First Search?

   Breadth-First Search lebih terorganisir karena memanfaatkan penggunaan matriks 3D dp, yang memungkinkan sebuah penyimpanan hasil sementara. Informasi di matriks ini digunakan kembali sehingga menghindari eksplorasi jalur yang sama secara berulang. Misal, jika matriks $dp[x][y][k]$ sudah dihitung, maka hasilnya tidak perlu dihitung ulang.

   Di sisi lainnya, Depth-First Search tidak menggunakan penyimpanan sementara, sehingga algoritma mengeksplorasi semua jalur tanpa menyimpan hasil sebelumnya. Akibatnya, Depth-First Search melakukan pekerjaan yang lebih signifikan, terutama untuk input size yang lebih besar, seperti yang didapatkan di tabel running time.

2. Mengapa Breadth-First Search lebih stabil dibandingkan dengan Depth-First Search di aplikasi?
   Seperti yang sudah dijelaskan sebelumnya, Breadth-First Search menggunakan matriks dp untuk menyimpan informasi setiap langkah dan posisi. Hal ini membuat algoritma ini menghasilkan hasil yang lebih konsisten dan stabil.

   Sedangkan Depth-First Search bergantung pada rekursi dan eksplorasi jalur secara mendalam yang dapat menyebabkan ketidakstabilan jika ada banyak jalur yang harus dieksplorasi.

3. Mengapa Breadth-First Search jauh lebih efisien dibandingkan dengan Depth-First Search?
   Pada Depth-First Search, algoritma mengeksplorasi semua kemungkinan jalur, yang didapatkan dari semua kombinasi posisi kuda di papan:

   $dx = \{-2, -2, -1, -1, 1, 1, 2, 2\}$, $dy = \{-1, 1, -2, 2, -2, 2, -1, 1\}$

   Sebaliknya, Breadth-First Search hanya mempertimbangkan jalur yang relevan karena algoritmanya bekerja berdasarkan informasi langkah sebelumnya yang disimpan di matriks.

8. Spesifikasi Alat

- Alat: Victus by HP Gaming Laptop 15-fa1788TX.
- CPU: i5-13420H, 2100Mhz, 8 Core(s), 12 Logical Processor(s).
- GPU: NVIDIA GeForce RTX 3050 6GB Laptop GPU.
- RAM: 8,00 GB (7,65 GB Total Physical Memory).
- Operating System: Windows 11 Home Single Language (23H2)
- System Type: 64-bit operating system, x64-based processor.

BAB III

Kesimpulan

BFS (Iteratif):

- BFS menggunakan metode iteratif dengan struktur data queue dan matriks 3D untuk menyimpan status setiap langkah.
- BFS lebih stabil dan efisien dalam memproses jalur karena menyimpan hasil langkah sebelumnya, sehingga tidak ada perhitungan ulang.
- BFS lebih cocok untuk menemukan jalur terpendek pada grid karena sifat eksplorasi level-ordernya.
- Kompleksitas waktu BFS berada di kisaran $O(k * N^2)$, di mana N adalah ukuran papan, dan k adalah jumlah langkah.

DFS (Rekursif):

- DFS bekerja dengan mengeksplorasi jalur sedalam mungkin terlebih dahulu (depth-order traversal).
- DFS tidak menyimpan informasi langkah sebelumnya sehingga semua jalur valid dihitung ulang.
- DFS lebih boros waktu untuk kasus ukuran besar, tetapi menggunakan lebih sedikit memori dibanding BFS.
- Kompleksitas waktu DFS adalah $O(8n)$, $O(8^n)$, karena mengeksplorasi semua jalur potensial.

Saran

Untuk kedepannya, diharapkan pengguna atau developer dapat mengimplementasikan algoritma Breadth-First Search dan Depth-First Search secara lebih efisien agar dapat menguji running time menggunakan input size yang lebih besar. Selain itu, karena bahasa pemrograman yang digunakan untuk studi kasus ini adalah Java, dan mungkin kurang efisien untuk mengelola dan memproses data besar, diharapkan di kedepannya developer dapat mengembangkan program di bahasa yang efisien, seperti C++.

DAFTAR PUSTAKA

Chess. “Terminologi Catur”. Chess. Diakses pada Desember 15, 2024. https://www.chess.com/id/terms/catur.

GeeksForGeeks. “Applications, Advantages and Disadvantages of Breadth First Search (BFS).” GeeksForGeeks. Diakses pada Desember 15, 2024. https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-breadth-first-search-bfs/.

GeeksForGeeks. “Breadth First Traversal (BFS) on a 2D array”. GeeksForGeeks. Diakses pada Desember 15, 2024. https://www.geeksforgeeks.org/breadth-first-traversal-bfs-on-a-2d-array/.

GeeksForGeeks. “Difference Between Depth First Search, Breadth First Search and Depth Limit Search in AI”. GeeksForGeeks. Diakses pada Desember 15, 2024. https://www.geeksforgeeks.org/difference-between-depth-first-search-breadth-first-search-and-depth-limit-search-in-ai/.

GeeksForGeeks. “Time and Space Complexity of Breadth First Search (BFS)”. GeeksForGeeks. Diakses pada Desember 15, 2024. https://www.geeksforgeeks.org/time-and-space-complexity-of-breadth-first-search-bfs/.

GeeksForGeeks. “Time and Space Complexity of Depth First Search (DFS)”. GeeksForGeeks. Diakses pada Desember 15, 2024. https://www.geeksforgeeks.org/time-and-space-complexity-of-depth-first-search-dfs/.

K. Pratama, Muhammad Ridho. “Mengenal dan Menghitung Time Complexity dan Space Complexity”. Medium. Diakses pada Desember 15, 2024. https://medium.com/99ridho/mengenal-dan-menghitung-time-complexity-dan-space-complexity-6418ea767336.