

**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

DAFTAR ISI.....	1
BAB I.....	2
1. Pendahuluan.....	2
2. Definisi dan Pseudocode.....	2
2.1. Breadth-First Search (BFS).....	2
2.2. Depth-First Search (DFS).....	3
2.3. Pseudocode Breadth-First Search (Iteratif).....	4
2.4. Pseudocode Depth-First Search (Rekursif).....	6
BAB II.....	9
3. Basic Operation.....	9
4. Perhitungan Kompleksitas Waktu.....	9
4.1. Worst Case Scenario.....	9
4.2. Best Case Scenario.....	10
4.3. Average Case Scenario.....	11
5. Notasi Asimtotik.....	12
5.1. Breadth-First Search.....	12
5.2. Depth-First Search.....	12
6. Kelas Efisiensi.....	12
6.1. Breadth-First Search.....	12
6.2. Depth-First Search.....	12
7. Visualisasi Tabel dan Grafik Running Time.....	13
8. Spesifikasi Alat.....	14
BAB III.....	16
Kesimpulan.....	16
Saran.....	16
DAFTAR PUSTAKA.....	17

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 $N \times N$ 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 menjamin 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)

```
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 $N \times N$.

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 Tujuan

Menghitung jumlah cara totalCara agar kuda catur dapat bergerak dari posisi awal (x,y) ke posisi tujuan(x2,y2) di papan catur berukuran n x n, menggunakan langkah rekursif dan mempertimbangkan semua kemungkinan gerakan kuda.

2.4.2 Pendekatan

DFS adalah teknik eksplorasi yang mencoba semua kemungkinan secara mendalam untuk menemukan solusi.

- Setiap langkah rekursi mengeksplorasi 8 kemungkinan gerakan kuda.
- Setiap jalur dihentikan jika
 - a. Langkah K habis
 - b. Posisi keluar dari batas papan catur.

Pendekatan ini cocok untuk menghitung jumlah semua kemungkinan cara, karena setiap jalur yang valid ditelusuri dan dihitung.

2.4.3 Langkah-langkah

1. Inisialisasi input

- Ukuran papan catur($n \times n$),
- Posisi awal kuda(x, y),
- Posisi tujuan(x_2, y_2),
- Jumlah langkah tersisa(K).

2. Periksa kondisi basis(Base case)

Jika $K = 0$

- Cek apakah posisi saat ini(x, y) sama dengan posisi tujuan(x_2, y_2):
 - Jika iya, kembalikan 1 (artinya ada satu cara mencapai tujuan).
 - Jika tidak, kembalikan 0 (tidak ada cara mencapai tujuan).

Jika $K < 0$

- Langkah sudah habis atau negatif, kembalikan 0 karena tidak mungkin mencapai tujuan

3. Rekursi untuk gerakan kuda

Jika langkah $K > 0$, lakukan langkah-langkah berikut:

1. Inisialisasi total cara: Buat variabel $\text{totalcara} = 0$ untuk menyimpan jumlah semua cara valid.
2. Iterasi untuk 8 gerakan kuda: kuda memiliki 8 kemungkinan gerakan dalam bentuk huruf "L". Gunakan dua array (dx dan dy) untuk mendefinisikan pergeseran posisi:

- $dx = [2, 2, -2, -2, 1, 1, -1, -1]$
- $dy = [1, -1, 1, -1, 2, -2, 2, -2]$ Setiap indeks i dari dx dan dy mewakili satu gerakan kuda.

3. Untuk setiap gerakan kuda

- Hitung posisi baru kuda:
$$\text{nextX} = x + dx[i], \text{nextY} = y + dy[i]$$
- Periksa apakah posisi baru valid (dalam batas papan catur):
 - Valid jika $0 \leq \text{nextX} < n$ dan $0 \leq \text{nextY} < n$.
- Jika posisi valid, panggil fungsi secara rekursif:

$\text{totalCara} += \text{HitungLangkahDFS_Rekursif}(n, \text{nextX}, \text{nextY}, x_2, y_2, k-1)$

- Artinya, lanjutkan pencarian dari posisi baru dengan langkah tersisa $k - 1$

4. Akumulasi dan kembalikan hasil

Setelah mengevaluasi semua 8 kemungkinan gerakan kuda, hasil akhirnya ada di variabel `totalCara`, yang menyimpan jumlah cara valid untuk mencapai posisi tujuan.

BAB II

3. Basic Operation

Pada algoritma Breadth-First Search, operasi dasarnya adalah *if* (*posisiBaruX* ≥ 0 && *posisiBaruX* $<$ *ukuranPapan* && *posisiBaruY* ≥ 0 && *posisiBaruY* $<$ *ukuranPapan*) *then*. 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 $N \times N$.

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. Worst Case Scenario

Worst Case Scenario menunjukkan worst case scenario dari kinerja algoritma. Worst Case dapat direpresentasikan sebagai big-O yang memberikan batas atas dari waktu running eksekusi sebuah algoritma. Big-O memiliki definisi formal: $O(f(n)) = \{g(n) \mid 0 \leq g(n) \leq cf(n)\}$ untuk semua $n \geq n_0$ untuk suatu konstanta c dan non-negative integer n_0 . Jika $g(n)$ merupakan anggota himpunan $O(f(n))$, kita tulis $g(n) \in O(f(n))$ atau $g(n) = O(f(n))$.

4.1.1. 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:

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

Dimana:

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

Sehingga:

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

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

4.1.2. 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:

$$TDFS = (\text{Jumlah Simpul}) \times (\text{Jumlah Sisi})$$

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

Jadi, worst case time complexity untuk DFS pada papan catur berukuran $N \times N$ adalah $O(N^4)$.

4.2. Best Case Scenario

Best Case Scenario menunjukkan best case scenario dari kinerja algoritma. Best Case dapat direpresentasikan sebagai big- Ω yang memberikan batas bawah dari waktu running eksekusi sebuah algoritma. Big- Ω memiliki definisi formal: $\Omega(f(n)) = \{g(n) \mid 0 \leq cf(n) \leq g(n)\}$ untuk semua $n \geq n_0$ untuk suatu konstanta c dan non-negative integer n_0 . Jika $g(n)$ merupakan anggota himpunan $\Omega(f(n))$, kita tulis $g(n) \in \Omega(f(n))$ atau $g(n) = \Omega(f(n))$.

4.2.1. Breadth-First Search

Pada Breadth-First Search, best case harus meliputi kondisi dimana posisi awal ($awalX$, $awalY$) sama dengan posisi tujuan ($tujuanX$, $tujuanY$) karena algoritma hanya perlu memeriksa langkah ke-0. Sehingga, worst best case dari algoritma ini adalah $O(1)$.

4.2.2. Depth-First Search

Pada Depth-First Search, best case terjadi ketika solusi ditemukan pada langkah pertama ($k = 0$) atau sangat cepat setelah beberapa langkah pertama, tanpa perlu mengeksplorasi banyak jalur.

4.3. Average Case Scenario

Average Case Scenario menunjukkan average case scenario dari kinerja algoritma. Average Case dapat direpresentasikan sebagai big- Θ yang memberikan batas tengah dari waktu running eksekusi sebuah algoritma. Big- Θ memiliki definisi formal: $\Theta(f(n)) = \{g(n) \mid c_1f(n) \leq g(n) \leq c_2f(n)\}$ untuk semua $n \geq n_0$ untuk suatu konstanta c_1 , c_2 dan non-negative integer n_0 . Jika $g(n)$ merupakan anggota himpunan $\Theta(f(n))$, kita tulis $g(n) \in \Theta(f(n))$ atau $g(n) = \Theta(f(n))$.

4.3.1. Breadth-First Search

Pada Breadth-First Search, average case harus meliputi kondisi dimana posisi tujuan dapat dicapai dalam beberapa langkah tertentu. Sehingga, average case dari algoritma ini adalah $O(k' * N^2)$.

4.3.2. Depth-First Search

Pada Depth-First Search, average case mengacu pada kondisi di mana algoritma mencapai tujuan dengan jumlah langkah yang lebih sedikit daripada kasus terburuk. Dalam kasus ini, DFS tidak akan menjelajahi semua kemungkinan jalur, melainkan berhenti lebih awal ketika menemukan solusi. Kompleksitas waktu rata-rata dapat dipengaruhi oleh faktor-faktor seperti kedalaman solusi dan struktur graf atau papan yang sedang dieksplorasi.

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 $O(8^k)$, 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 $O(k * N^2)$.

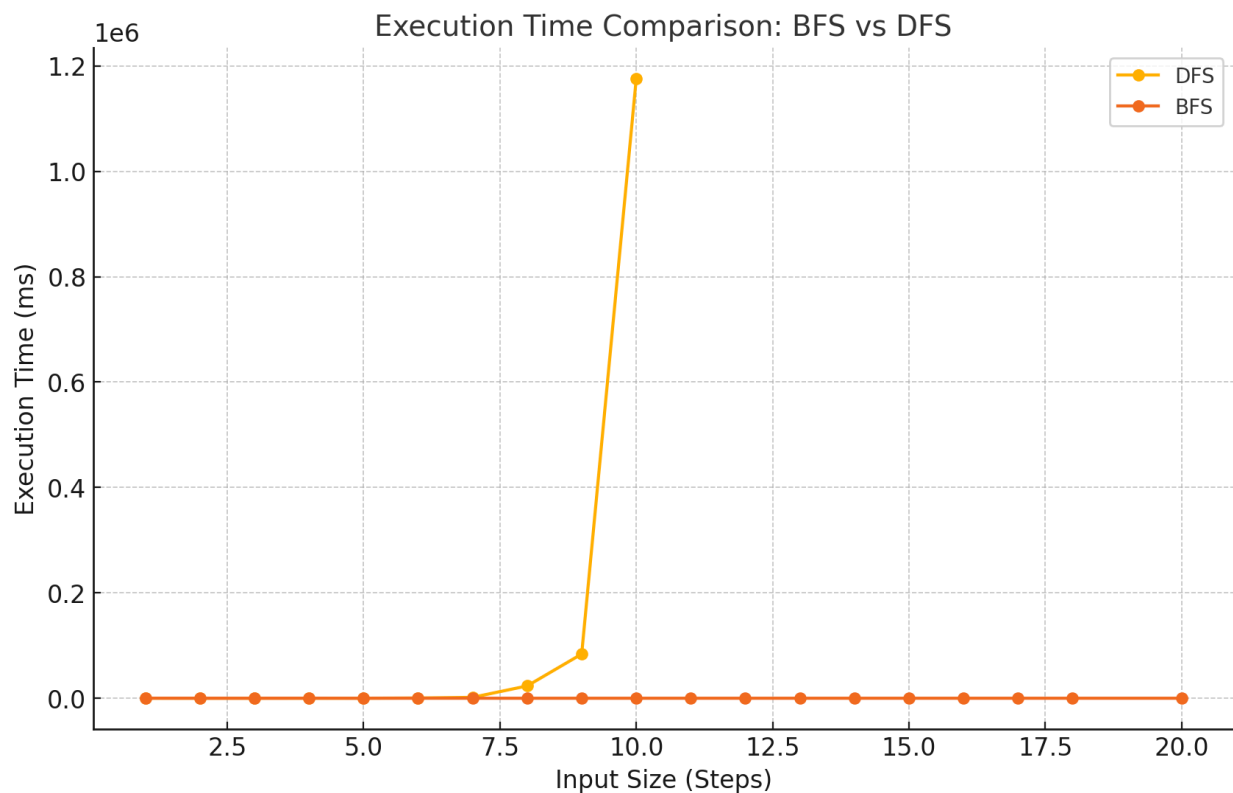
6.2. Depth-First Search

Algoritma ini memiliki kompleksitas waktu $O(8^k)$, 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	1739ms	23364ms	83915ms	1175644ms

Tabel 7.1.1.



Gambar 7.1.1.

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 $O(8k)$, $O(8^k)$, 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>.