Laporan Tugas Kecil IF2211 Strategi Algoritma

Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding

Disusun Oleh:

Qodri Azkarayan (13523010)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

Daftar Isi

Bab 1

1.1 Deskripsi Masalah

Permainan Rush Hour merupakan salah satu puzzle logika berbasis grid yang menantang pemain untuk memindahkan kendaraan-kendaraan di dalam papan permainan agar kendaraan utama (primary piece) dapat keluar dari kemacetan melalui pintu keluar yang tersedia. Permainan ini memiliki kompleksitas yang cukup tinggi karena keterbatasan gerakan setiap kendaraan, yang hanya bisa bergeser secara linear sesuai orientasinya tanpa dapat berputar atau melewati kendaraan lain.

Permasalahan penyelesaian Rush Hour secara optimal dapat dimodelkan sebagai permasalahan pencarian jalur (pathfinding) dalam ruang keadaan yang luas dan kompleks. Oleh karena itu, diperlukan algoritma pencarian yang efisien untuk menemukan solusi dengan jumlah langkah minimum. Algoritma pencarian seperti Uniform Cost Search (UCS), Greedy Best First Search (GBFS), dan A* merupakan metode yang umum digunakan dalam menyelesaikan permasalahan pathfinding dengan berbagai pendekatan dan efisiensi.

Bab 2

2.1 Algoritma Path Finding

2.1.1 Uniform Cost Search

Uniform Cost Search adalah algoritma pencarian jalur yang memperluas node berdasarkan biaya terendah kumulatif dari simpul awal sampai simpul tersebut. UCS merupakan varian dari Breadth-First Search yang mempertimbangkan bobot atau biaya pada setiap langkah.

UCS menggunakan fungsi evaluasi:

$$f(n) = g(n)$$

dengan $g(n)$ adalah biaya jalur terkecil dari node awal ke node n n. Node dengan nilai $g(n)$ terkecil akan diperluas terlebih dahulu.

Implementasi algoritma UCS dalam permainan Rush Hour adalah algoritma menyimpan semua konfigurasi yang mungkin ke dalam sebuah List dan menandai konfigurasi yang sudah dikunjungi dalam sebuah Set agar tidak diproses ulang. Dari setiap State, algoritma menghasilkan semua langkah valid yang dapat dilakukan, kemudian memperbarui konfigurasi papan dan posisi kendaraan sesuai langkah tersebut. Setiap State baru ditambahkan ke antrean dan diurutkan berdasarkan biaya kumulatif langkah yang sudah diambil. Proses ini berlanjut hingga kendaraan utama mencapai tujuan atau antrean kosong.

2.1.2 Greedy Best First Search

Greedy Best First Search merupakan algoritma pencarian yang menggunakan heuristik untuk memilih node yang tampaknya paling dekat dengan tujuan tanpa memperhitungkan biaya jalur sebenarnya.

GBFS menggunakan fungsi evaluasi:

$$f(n) = h(n)$$

dengan $h(n)$ adalah estimasi biaya dari node n menuju tujuan. Algoritma ini memilih node dengan nilai heuristik terkecil untuk diperluas.

Algoritma GBFS pada kode ini bekerja dengan memperluas simpul yang memiliki nilai heuristik terkecil terlebih dahulu untuk mencapai tujuan pada puzzle Rush Hour. Algoritma memulai dari keadaan awal, menyimpan konfigurasi papan, posisi kendaraan, serta langkah-langkah yang telah diambil ke dalam sebuah List. Pada setiap iterasi, algoritma mengambil State dengan nilai heuristik terendah dari List, kemudian memeriksa apakah konfigurasi tersebut sudah pernah dikunjungi dengan sebuah Set agar tidak terjadi pengulangan. Jika belum, algoritma menandainya sebagai sudah dikunjungi dan menghasilkan semua langkah valid dari keadaan tersebut. Setiap langkah valid akan menghasilkan konfigurasi papan baru dengan posisi kendaraan yang diperbarui, dan nilai heuristik untuk konfigurasi baru tersebut dihitung menggunakan salah satu dari dua metode heuristik yang dipilih yaitu

berdasarkan jumlah kendaraan penghalang di jalur kendaraan utama atau jarak kendaraan utama ke pintu keluar. Konfigurasi baru beserta nilai heuristiknya dimasukkan kembali ke List. List kemudian diurutkan berdasarkan nilai heuristik agar State yang memiliki heuristik terkecil diperiksa terlebih dahulu. Proses ini berlangsung hingga kendaraan utama mencapai posisi tujuan atau antrean habis.

2.1.3 A*

A* adalah algoritma pencarian jalur yang menggabungkan keunggulan UCS dan GBFS dengan menggunakan fungsi evaluasi yang mempertimbangkan biaya jalur sejauh ini dan estimasi biaya ke tujuan.

A* menggunakan fungsi evaluasi:

$$f(n) = g(n) + h(n)$$

dengan $g(n)$ adalah biaya jalur dari node awal ke node n. $h(n)$ adalah heuristik atau estimasi biaya terendah dari node n menuju tujuan.

Algoritma A* pada game Rush Hour dimulai dengan menyimpan keadaan awal permainan, meliputi konfigurasi papan, posisi kendaraan, serta daftar langkah yang sudah diambil ke dalam sebuah antrean. Pada setiap iterasi, algoritma mengambil simpul dengan nilai evaluasi f(n) terendah dari List, yang merupakan gabungan antara biaya sebenarnya untuk mencapai keadaan tersebut dan nilai heuristik sebagai perkiraan biaya menuju tujuan.

Jika State tersebut belum pernah dikunjungi sebelumnya, algoritma menandainya sebagai sudah diperiksa agar tidak memproses ulang konfigurasi yang sama. Selanjutnya, algoritma menghasilkan semua langkah valid dari konfigurasi saat ini dan membuat salinan baru dari papan serta posisi kendaraan yang diperbarui berdasarkan langkah tersebut. Nilai heuristik untuk konfigurasi baru dihitung dengan menambahkan satu langkah ke nilai heuristik keadaan sebelumnya serta menambahkan nilai dari salah satu heuristik yang dipilih, yaitu jumlah kendaraan penghalang di jalur kendaraan utama atau jarak kendaraan utama ke pintu keluar. Konfigurasi baru berikut nilai heuristiknya ini kemudian dimasukkan ke dalam List.

Antrean disortir ulang berdasarkan nilai evaluasi $f(n)$ sehingga simpul dengan estimasi biaya total terkecil diprioritaskan untuk diperluas. Proses ini diulang hingga kendaraan utama mencapai posisi tujuan pada papan.

2.2 Analisis Algoritma Path Finding

2.2.1 Definisi $f(n)$ dan $g(n)$

Dalam konteks algoritma pencarian jalur, $g(n)$ adalah biaya kumulatif yang telah dikeluarkan untuk mencapai simpul n dari titik awal. Pada permainan Rush Hour, $g(n)$ dapat diartikan sebagai jumlah langkah atau perpindahan kendaraan yang sudah dilakukan untuk sampai pada konfigurasi papan saat ini. Sedangkan

$f(n)$ adalah fungsi evaluasi yang merupakan perkiraan total biaya jalur dari titik awal sampai tujuan melalui simpul n.

2.2.2 Heuristik yang *admissible*

Heuristik disebut admissible jika tidak pernah melebih-lebihkan biaya sebenarnya dari suatu simpul menuju tujuan, atau secara formal $h(n) \leq h^*(n)$, dengan $h^*(n)$ adalah biaya sebenarnya ke tujuan. Heuristik jarak kendaraan utama ke pintu keluar dan jumlah mobil penghalang pada puzzle Rush Hour dianggap admissible karena keduanya memberikan estimasi biaya yang tidak melebihi biaya sebenarnya untuk mencapai tujuan. Jarak kendaraan utama ke pintu keluar menghitung langkah minimum yang diperlukan jika tidak ada penghalang, sehingga nilainya selalu lebih kecil atau sama dengan biaya nyata dalam kondisi sebenarnya. Sedangkan jumlah mobil penghalang hanya menghitung jumlah kendaraan yang harus digeser agar jalan terbuka, tanpa mempertimbangkan berapa kali setiap kendaraan harus digeser, sehingga juga tidak melebih-lebihkan biaya nyata. Karena keduanya memberikan estimasi yang optimis dan tidak berlebihan, heuristik ini memenuhi definisi admissible, memastikan algoritma A* tetap dapat menemukan solusi optimal pada permainan Rush Hour.

2.2.3 UCS vs BFS pada Rush Hour

Secara umum, Uniform Cost Search (UCS) dan Breadth-First Search (BFS) berbeda karena UCS mempertimbangkan biaya langkah, sedangkan BFS hanya mempertimbangkan jumlah langkah tanpa bobot. Namun, jika setiap langkah pada Rush Hour dianggap memiliki biaya yang sama (misalnya, satu langkah per perpindahan kendaraan), maka UCS dan BFS akan berperilaku serupa, memperluas node secara berurutan berdasarkan kedalaman pohon pencarian. Dalam kasus tersebut, urutan node yang dibangkitkan dan path yang dihasilkan oleh UCS dan BFS akan sama dan menghasilkan solusi dengan jumlah langkah minimal.

2.2.4 A* vs UCS

Secara teoritis, algoritma A* lebih efisien daripada UCS ketika heuristik yang digunakan adalah admissible dan baik. A* menggunakan fungsi evaluasi $f(n) = g(n) + h(n)$ yang tidak hanya mempertimbangkan biaya jalur yang telah ditempuh ($g(n)$), tetapi juga perkiraan biaya tersisa ke tujuan ($h(n)$). Hal ini memungkinkan A* untuk mengarahkan pencarian ke jalur yang lebih menjanjikan dan menghindari eksplorasi simpul yang kurang relevan. Sebaliknya, UCS hanya mempertimbangkan $g(n)$ tanpa informasi ke depan, sehingga dapat mengeksplorasi lebih banyak simpul. Oleh karena itu, pada Rush

Hour, A* biasanya membutuhkan lebih sedikit ekspansi node dan waktu komputasi dibanding UCS, sehingga lebih efisien.

2.2.5 Keandalan GBFS

Greedy Best First Search (GBFS) memilih simpul yang menurut heuristiknya paling dekat dengan tujuan tanpa memperhatikan biaya jalur yang sudah ditempuh. Karena itu, GBFS tidak menjamin solusi optimal pada Rush Hour. Algoritma ini berisiko terjebak pada jalur yang tampaknya cepat mendekati tujuan secara heuristik, tetapi sebenarnya bukan jalur terbaik dalam hal jumlah langkah. Selain itu, GBFS dapat terperangkap dalam local minima atau plateau sehingga tidak menyelesaikan pencarian dengan benar. Dengan demikian, meskipun GBFS bisa menemukan solusi dengan cepat, solusi tersebut belum tentu optimal.

Bab 3

3.1 Deskripsi Program

Program ini ditulis menggunakan bahasa pemrograman Java. Program memanfaatkan berbagai kelas dan struktur data bawaan Java seperti Scanner untuk input, HashMap dan Map untuk penyimpanan data dinamis, serta LinkedList dan List untuk pengelolaan urutan langkah dan status pencarian. Struktur Set seperti HashSet digunakan untuk menyimpan status yang sudah pernah dikunjungi agar menghindari eksplorasi ulang.

3.2 Source Code

Source Code dapat diakses pada tautan berikut:
https://github.com/qodriazka/Tucil3_13523010

3.2.1 Main.java

```java
import java.util.Scanner;
import java.util.HashMap;
import java.util.Map;
import java.util.LinkedList;
import java.util.List;
import java.util.HashSet;
import java.util.Set;
import java.io.File;
import javax.sound.sampled.*;

public class Main {
    static int counter = 0;
    public static void main(String[] args) {
        File file = new File("resource\\rushHour.wav");
        try (AudioInputStream audio = AudioSystem.getAudioInputStream(file)){
            Clip clip = AudioSystem.getClip();
            clip.open(audio);
            clip.loop(Clip.LOOP_CONTINUOUSLY);
            clip.start();
        }catch (Exception e) {
            System.out.println("Something went wrong with audio player");
        }
        System.out.print("\033c");
        System.out.println(
        "┬ ┬┌─┐┬  ┌─┐┌─┐┌┬┐┌─┐  ┌┬┐┌─┐\n" +
        "│││├┤ │  │  │ ││││├┤    │ │ │\n" +
        "└┴┘└─┘┴─┘└─┘└─┘┴ ┴└─┘   ┴ └─┘"
        );
        System.out.println("888~-_                      888           888   |
,d88~~\\          888                              d8b      ,d  d8b      ,d ");
        System.out.println("888   \\  888  888  d88~\\ 888-~88e      888___|  e88~-_
888  888 888-~\\      8888      e88~-_  888 Y88b    /  e88~~8e  888-~\\ !Y88! ,d888 !Y88!
,d888 ");
        System.out.println("888    | 888  888 C888   888  888      888   | d888   i 888
888 888         `Y88b   d888   i 888  Y88b  /  d888  88b 888     Y8Y    888  Y8Y    888
");
```

```
        System.out.println("888   /  888  888  Y88b  888  888      888   | 8888   | 888
888 888          `Y88b, 8888   | 888   Y88b/   8888__888 888      8     888   8     888
");
        System.out.println("888_-~   888  888   888D 888  888      888   | Y888   ' 888
888 888            8888 Y888   ' 888    Y8/    Y888    , 888      e     888   e     888
");
        System.out.println("888 ~-_  \"88_-888 \\_88P  888  888      888   |  \"88_-~
\"88_-888 888         \\__88P'  \"88_-~  888     Y      \"88___/  888     \"8\"    888
\"8\"    888 ");

        Scanner input = new Scanner(System.in);
        System.out.print("Masukkan nama file txt yang ada di folder test (misal:
input1): ");
        String filename = input.nextLine();
        File inputFile = new File("..\\test\\" + filename + ".txt");

        while (!validateInputFile(inputFile)) {
            System.out.println("File tidak ditemukan atau format salah. Silakan masukkan
nama file yang benar.");
            System.out.print("Masukkan nama file txt yang ada di folder test (misal:
input1): ");
            filename = input.nextLine();
            inputFile = new File("..\\test\\" + filename + ".txt");
        }

        int A = 0, B = 0, N = 0;
        char[][] papan = null;
        int tujuanKolom = -1;
        int tujuanBaris = -1;
        Map<Character, Map<String, Integer>> huruf = new HashMap<>();

        try (Scanner fileScanner = new Scanner(inputFile)) {
            if (fileScanner.hasNextLine()) {
            String[] ab = fileScanner.nextLine().trim().split("\\s+");
            A = Integer.parseInt(ab[0]);
            B = Integer.parseInt(ab[1]);
            }
            if (fileScanner.hasNextLine()) {
            N = Integer.parseInt(fileScanner.nextLine().trim());
            }
            papan = new char[A][B];
            String baris;
            for (int i = 0; i < A; i++) {
            if (!fileScanner.hasNextLine()) break;
            baris = fileScanner.nextLine();
            int indeks = 0;
            if (baris.length() < B || (baris.charAt(0) == ' ' && baris.charAt(1) == '
')) {
                tujuanBaris = 0;
                for (int j = 0; j < baris.length(); j++) {
                if (baris.charAt(j) == 'X') {
                    tujuanKolom = j;
```

```
                }
                }
                if (fileScanner.hasNextLine()) baris = fileScanner.nextLine();
            } else if (baris.length() > B) {
                if (baris.charAt(0) == ' ') {
                indeks = 1;
                } else {
                tujuanBaris = i;
                if (baris.charAt(0) == 'X') {
                    tujuanKolom = 0;
                    indeks = 1;
                } else {
                    tujuanKolom = B - 1;
                }
                }
            }
            for (int j = indeks; j < indeks + B; j++) {
                char x = baris.charAt(j);
                papan[i][j - indeks] = x;
                if (x != '.' && !huruf.containsKey(x)) {
                Map<String, Integer> detail = new HashMap<>();
                detail.put("baris", i);
                detail.put("kolom", j - indeks);
                detail.put("panjang", 2); //set defaultnya 2
                if (j + 1 < indeks + B && baris.charAt(j + 1) == x) {
                    detail.put("orientasi", 1); // 1 kalo horizontal, 2 kalo vertikal
                    if (j + 2 < indeks + B && baris.charAt(j + 2) == x) {
                    detail.put("panjang", 3);
                    }
                } else {
                    detail.put("orientasi", 2);
                }
                huruf.put(x, detail);
                }
            }
            }
            if (tujuanBaris == -1 && fileScanner.hasNextLine()) {
            tujuanBaris = A - 1;
            baris = fileScanner.nextLine();
            for (int j = 0; j < baris.length(); j++) {
                if (baris.charAt(j) == 'X') {
                tujuanKolom = j;
                }
            }
            }
        } catch (Exception e) {
            System.out.println("Terjadi kesalahan saat membaca file input.");
            input.close();
            return;
        }
        System.out.println("A: " + A + ", B: " + B + ", N: " + N);
```

```
        String pilihan = "4";
        while (!pilihan.equals("1") && !pilihan.equals("2") && !pilihan.equals("3")) {
            System.out.println("Pilih Algoritma Pencarian");
            System.out.println("1. Uniform Cost Search");
            System.out.println("2. Greedy Best First Search");
            System.out.println("3. A*");
            pilihan = input.nextLine();
        }
        String heu = "3";
        if (pilihan.equals("2") || pilihan.equals("3")) {
            while (!heu.equals("1") && !heu.equals("2")) {
            System.out.println("Pilih Heuristik");
            System.out.println("1. Jumlah mobil yang menghalangi");
            System.out.println("2. Jarak mobil ke pintu keluar");
            heu = input.nextLine();
            }
        }
        // cek panjang piece vertikal, ubah kalo ternyata panjangnya 3
        for (Map.Entry<Character, Map<String, Integer>> entry : huruf.entrySet()) {
            Map<String, Integer> detail = entry.getValue();
            Integer orientasi = detail.get("orientasi");
            if (orientasi == 2) {
                int row = detail.get("baris");
                int col = detail.get("kolom");
                if(row+2 < B && papan[row][col]==papan[row+2][col]){
                    int panjang = detail.get("panjang");
                    detail.put("panjang", panjang+1);
                }
            }
        }
        counter = 0;
        long start = System.currentTimeMillis();
        List<Langkah> langkah = new LinkedList<>();
        if(pilihan.equals("1")){
            System.out.println("Hasil pencarian UCS");
            langkah = new Main().UCS(papan, huruf, tujuanBaris, tujuanKolom, A, B);
        }else if(pilihan.equals("2")){
            System.out.println("Hasil pencarian GBFS");
            langkah = new Main().GBFS(papan, huruf, tujuanBaris, tujuanKolom, A, B,
heu);
        }else if(pilihan.equals("3")){
            System.out.println("Hasil pencarian A*");
            langkah = new Main().AStar(papan, huruf, tujuanBaris, tujuanKolom, A, B,
heu);
        }
        if(langkah.size() == 0){
            System.out.println("Tidak ditemukan solusi (hapus spasi di akhir tiap baris
di file input)");
            input.close();
            return;
        }
        long end = System.currentTimeMillis();
```

```
        StringBuilder output = new StringBuilder()
                .append("Papan awal\n")
                .append(cetakPapan(papan, -1, -1, 0, 1));
        int gerakan = 1;
        for (Langkah l : langkah) {
            output.append(System.lineSeparator());
            new Main().ubahPapan(papan, l, huruf);
            String arah = "";
            if(huruf.get(l.piece).get("orientasi") == 1){
                if(l.jarak > 0){
                    arah = l.piece + "-kanan " + l.jarak;
                }else if(l.jarak < 0){
                    arah = l.piece + "-kiri " + Math.abs(l.jarak);
                }
            }else{
                if(l.jarak > 0){
                    arah = l.piece + "-bawah " + l.jarak;
                }else if(l.jarak < 0){
                    arah = l.piece + "-atas " + Math.abs(l.jarak);
                }
            }
            output.append("Gerakan ke-" + gerakan + ": " + arah);
            output.append(System.lineSeparator());
            output.append(cetakPapan(papan, huruf.get(l.piece).get("baris"),
huruf.get(l.piece).get("kolom"), huruf.get(l.piece).get("panjang"),
huruf.get(l.piece).get("orientasi")));
            gerakan++;
        }
        output.append(System.lineSeparator());
        System.out.println(output.toString());
        System.out.println("Waktu eksekusi: " + (end - start) + " ms");
        System.out.println("Jumlah state yang dicari: " + counter);
        System.out.println("Apakah ingin menyimpan hasil ke file? (y/n)");
        String save = input.nextLine();
        if (save.equalsIgnoreCase("y")) {
            System.out.print("Masukkan nama file output (tanpa ekstensi): ");
            String outputFilename = input.nextLine();
            File outputFile = new File("..\\test\\" + outputFilename + ".txt");
            String cleanedOutput = output.toString().replaceAll("\u001B\\[[;\\d]*m",
"");

            try (java.io.PrintWriter writer = new java.io.PrintWriter(outputFile)) {
                writer.println(cleanedOutput);
                System.out.println("Hasil telah disimpan ke " +
outputFile.getAbsolutePath());
            } catch (Exception e) {
                System.out.println("Terjadi kesalahan saat menyimpan file.");
            }
        }

        input.close();
    }
```

```
    public static boolean validateInputFile(File file) {
        if (!file.exists() || !file.isFile()) return false;
        try (Scanner sc = new Scanner(file)) {
            if (!sc.hasNextLine()) return false;
            String[] ab = sc.nextLine().trim().split("\\s+");
            if (ab.length != 2) return false;
            int A = Integer.parseInt(ab[0]);
            if (!sc.hasNextLine()) return false;
            int lineCount = 0;
            while (sc.hasNextLine() && lineCount < A) {
                sc.nextLine();
                lineCount++;
            }
            return lineCount == A;
        } catch (Exception e) {
            return false;
        }
    }

    public void ubahPapan(char[][] papan, Langkah l, Map<Character, Map<String,
Integer>> huruf){
        int barisLama = huruf.get(l.piece).get("baris");
        int kolomLama = huruf.get(l.piece).get("kolom");
        if(huruf.get(l.piece).get("orientasi") == 1){
            for(int i = 0; i<huruf.get(l.piece).get("panjang"); i++){
                papan[barisLama][kolomLama+i] = '.';
            }
            for (int i = 0; i<huruf.get(l.piece).get("panjang"); i++) {
                papan[barisLama][kolomLama+i+l.jarak] = l.piece;
            }
            huruf.get(l.piece).put("kolom", kolomLama + l.jarak);
        }else{
            for(int i = 0; i<huruf.get(l.piece).get("panjang"); i++){
                papan[barisLama+i][kolomLama] = '.';
            }
            for (int i = 0; i < huruf.get(l.piece).get("panjang"); i++) {
                papan[barisLama+i+l.jarak][kolomLama] = l.piece;
            }
            huruf.get(l.piece).put("baris", barisLama + l.jarak);
        }
    }

    public List<Langkah> AStar(char[][] papan, Map<Character, Map<String, Integer>>
huruf, int tujuanBaris, int tujuanKolom, int A, int B, String heuristik) {
        List<State> tree = new LinkedList<>();
        List<Langkah> langkah = new LinkedList<>();
        Set<String> visited = new HashSet<>();
        State awal = new State(papan, 0, huruf, langkah, visited);
        tree.add(awal);
        while(finish(tree.get(0).pieces, tujuanBaris, tujuanKolom) == false){
            State current = tree.get(0);
```

```
            tree.remove(0);
            Map<Character, Map<String, Integer>> pieces = current.pieces;
            String visitedString = hurufToString(current.pieces);
            if(!current.visited.contains(visitedString)){
                current.visited.add(visitedString);
                List<Langkah> langkahBaru = possibleMove(current.board, current.pieces,
A, B);
                for (Langkah l : langkahBaru) {
                    Map<Character, Map<String, Integer>> piecesBaru = new HashMap<>();
                    for (Map.Entry<Character, Map<String, Integer>> entry :
current.pieces.entrySet()) {
                        Map<String, Integer> innerMap = new HashMap<>();
                        for (Map.Entry<String, Integer> innerEntry :
entry.getValue().entrySet()) {
                            innerMap.put(innerEntry.getKey(), innerEntry.getValue());
                        }
                        piecesBaru.put(entry.getKey(), innerMap);
                    }
                    int kolomLama = pieces.get(l.piece).get("kolom");
                    int barisLama = pieces.get(l.piece).get("baris");
                    List<Langkah> step = new LinkedList<>(current.steps);
                    step.add(l);
                    char[][] boardBaru = new char[A][B];
                    for (int i = 0; i < A; i++) {
                        System.arraycopy(current.board[i], 0, boardBaru[i], 0, B);
                    }
                    if(piecesBaru.get(l.piece).get("orientasi") == 1){
                        for(int i = 0; i<current.pieces.get(l.piece).get("panjang");
i++){
                            boardBaru[barisLama][kolomLama+i] = '.';
                        }
                        for (int i = 0; i<current.pieces.get(l.piece).get("panjang");
i++) {
                            boardBaru[barisLama][kolomLama+i+l.jarak] = l.piece;
                        }
                    }else{
                        for(int i = 0; i<current.pieces.get(l.piece).get("panjang");
i++){
                            boardBaru[barisLama+i][kolomLama] = '.';
                        }
                        for (int i = 0; i < piecesBaru.get(l.piece).get("panjang"); i++)
{
                            boardBaru[barisLama+i+l.jarak][kolomLama] = l.piece;
                        }
                    }
                    if(piecesBaru.get(l.piece).get("orientasi") == 1){
                        piecesBaru.get(l.piece).put("kolom", kolomLama + l.jarak);
                    }else{
                        piecesBaru.get(l.piece).put("baris", barisLama + l.jarak);
                    }
                    int heuristikValue  = current.heuristic;
                    if(heuristik.equals("1")){
```

```
                        heuristikValue += mobilPenghalang(boardBaru, piecesBaru,
tujuanBaris, tujuanKolom) + 1;
                    }else if(heuristik.equals("2")){
                        heuristikValue += jarak(piecesBaru, tujuanBaris, tujuanKolom) +
1;
                    }
                    State nextState = new State(boardBaru, heuristikValue, piecesBaru,
step, current.visited);
                    counter++;
                    tree.add(nextState);
                }
            }
            tree.sort((s1, s2) -> Integer.compare(s1.heuristic, s2.heuristic));
        }
        return tree.get(0).steps;
    }

    public List<Langkah> GBFS(char[][] papan, Map<Character, Map<String, Integer>>
huruf, int tujuanBaris, int tujuanKolom, int A, int B, String heuristik) {
        List<State> tree = new LinkedList<>();
        List<Langkah> langkah = new LinkedList<>();
        Set<String> visited = new HashSet<>();
        State awal = new State(papan, 0, huruf, langkah, visited);
        tree.add(awal);
        counter++;
        while(finish(tree.get(0).pieces, tujuanBaris, tujuanKolom) == false){
            State current = tree.get(0);
            tree.remove(0);
            Map<Character, Map<String, Integer>> pieces = current.pieces;
            String visitedString = hurufToString(current.pieces);
            if(!current.visited.contains(visitedString)){
                current.visited.add(visitedString);
                List<Langkah> langkahBaru = possibleMove(current.board, current.pieces,
A, B);
                for (Langkah l : langkahBaru) {
                    Map<Character, Map<String, Integer>> piecesBaru = new HashMap<>();
                    for (Map.Entry<Character, Map<String, Integer>> entry :
current.pieces.entrySet()) {
                        Map<String, Integer> innerMap = new HashMap<>();
                        for (Map.Entry<String, Integer> innerEntry :
entry.getValue().entrySet()) {
                            innerMap.put(innerEntry.getKey(), innerEntry.getValue());
                        }
                        piecesBaru.put(entry.getKey(), innerMap);
                    }
                    int kolomLama = pieces.get(l.piece).get("kolom");
                    int barisLama = pieces.get(l.piece).get("baris");
                    List<Langkah> step = new LinkedList<>(current.steps);
                    step.add(l);
                    char[][] boardBaru = new char[A][B];
                    for (int i = 0; i < A; i++) {
                        System.arraycopy(current.board[i], 0, boardBaru[i], 0, B);
```

```
                    }
                    if(piecesBaru.get(l.piece).get("orientasi") == 1){
                        for(int i = 0; i<current.pieces.get(l.piece).get("panjang");
i++){
                            boardBaru[barisLama][kolomLama+i] = '.';
                        }
                        for (int i = 0; i<current.pieces.get(l.piece).get("panjang");
i++) {
                            boardBaru[barisLama][kolomLama+i+l.jarak] = l.piece;
                        }
                    }else{
                        for(int i = 0; i<current.pieces.get(l.piece).get("panjang");
i++){
                            boardBaru[barisLama+i][kolomLama] = '.';
                        }
                        for (int i = 0; i < piecesBaru.get(l.piece).get("panjang"); i++)
{
                            boardBaru[barisLama+i+l.jarak][kolomLama] = l.piece;
                        }
                    }
                    if(piecesBaru.get(l.piece).get("orientasi") == 1){
                        piecesBaru.get(l.piece).put("kolom", kolomLama + l.jarak);
                    }else{
                        piecesBaru.get(l.piece).put("baris", barisLama + l.jarak);
                    }
                    int heuristikValue  = current.heuristic;
                    if(heuristik.equals("1")){
                        heuristikValue += mobilPenghalang(boardBaru, piecesBaru,
tujuanBaris, tujuanKolom);
                    }else if(heuristik.equals("2")){
                        heuristikValue += jarak(piecesBaru, tujuanBaris, tujuanKolom);
                    }
                    State nextState = new State(boardBaru, heuristikValue, piecesBaru,
step, current.visited);
                    tree.add(nextState);
                    counter++;
                }
            }
            tree.sort((s1, s2) -> Integer.compare(s1.heuristic, s2.heuristic));
            if(tree.size()==0){
                return new LinkedList<>();
            }
        }
        return tree.get(0).steps;
    }

    public List<Langkah> UCS(char[][] papan, Map<Character, Map<String, Integer>> huruf,
int tujuanBaris, int tujuanKolom, int A, int B) {
        List<State> tree = new LinkedList<>();
        List<Langkah> langkah = new LinkedList<>();
        Set<String> visited = new HashSet<>();
        State awal = new State(papan, 0, huruf, langkah, visited);
```

```
        tree.add(awal);
        counter++;
        while(finish(tree.get(0).pieces, tujuanBaris, tujuanKolom) == false){
            State current = tree.get(0);
            tree.remove(0);
            Map<Character, Map<String, Integer>> pieces = current.pieces;
            String visitedString = hurufToString(current.pieces);
            if(!current.visited.contains(visitedString)){
                current.visited.add(visitedString);
                List<Langkah> langkahBaru = possibleMove(current.board, current.pieces,
A, B);
                for (Langkah l : langkahBaru) {
                    Map<Character, Map<String, Integer>> piecesBaru = new HashMap<>();
                    for (Map.Entry<Character, Map<String, Integer>> entry :
current.pieces.entrySet()) {
                        Map<String, Integer> innerMap = new HashMap<>();
                        for (Map.Entry<String, Integer> innerEntry :
entry.getValue().entrySet()) {
                            innerMap.put(innerEntry.getKey(), innerEntry.getValue());
                        }
                        piecesBaru.put(entry.getKey(), innerMap);
                    }
                    int kolomLama = pieces.get(l.piece).get("kolom");
                    int barisLama = pieces.get(l.piece).get("baris");
                    List<Langkah> step = new LinkedList<>(current.steps);
                    step.add(l);
                    char[][] boardBaru = new char[A][B];
                    for (int i = 0; i < A; i++) {
                        System.arraycopy(current.board[i], 0, boardBaru[i], 0, B);
                    }
                    if(piecesBaru.get(l.piece).get("orientasi") == 1){
                        for(int i = 0; i<current.pieces.get(l.piece).get("panjang");
i++){
                            boardBaru[barisLama][kolomLama+i] = '.';
                        }
                        for (int i = 0; i<current.pieces.get(l.piece).get("panjang");
i++) {
                            boardBaru[barisLama][kolomLama+i+l.jarak] = l.piece;
                        }
                    }else{
                        for(int i = 0; i<current.pieces.get(l.piece).get("panjang");
i++){
                            boardBaru[barisLama+i][kolomLama] = '.';
                        }
                        for (int i = 0; i < piecesBaru.get(l.piece).get("panjang"); i++)
{
                            boardBaru[barisLama+i+l.jarak][kolomLama] = l.piece;
                        }
                    }
                    if(piecesBaru.get(l.piece).get("orientasi") == 1){
                        piecesBaru.get(l.piece).put("kolom", kolomLama + l.jarak);
                    }else{
```

```
                    piecesBaru.get(l.piece).put("baris", barisLama + l.jarak);
                }
                int heuristikValue  = current.heuristic+1;
                State nextState = new State(boardBaru, heuristikValue, piecesBaru,
step, current.visited);
                tree.add(nextState);
                counter++;
            }
        }
        tree.sort((s1, s2) -> Integer.compare(s1.heuristic, s2.heuristic));
        if(tree.size()==0){
            return new LinkedList<>();
        }
    }
    return tree.get(0).steps;
}

// Convert map huruf ke string biar bisa dicek udah dikunjungi atau belum
public String hurufToString(Map<Character, Map<String, Integer>> huruf){
    StringBuilder sb = new StringBuilder();
    for (Map.Entry<Character, Map<String, Integer>> entry : huruf.entrySet()) {
        Character hurufKey = entry.getKey();
        Map<String, Integer> detail = entry.getValue();
        sb.append(hurufKey);
        sb.append(detail.get("baris"));
        sb.append(detail.get("kolom"));
    }
    return sb.toString();
}

// Return langkah yang mungkin dalam bentuk List (huruf, jarak)
public List<Langkah> possibleMove(char[][] papan, Map<Character, Map<String,
Integer>> huruf, int A, int B){
    List<Langkah> langkah = new LinkedList<>();
    for (Map.Entry<Character, Map<String, Integer>> entry : huruf.entrySet()) {
        Character hurufKey = entry.getKey();
        Map<String, Integer> detail = entry.getValue();
        int baris = detail.get("baris");
        int kolom = detail.get("kolom");
        int panjang = detail.get("panjang");
        int orientasi = detail.get("orientasi");
        int i =  1;
        if(orientasi == 1){
            while(kolom+panjang+i-1 < B && papan[baris][kolom+panjang+i-1] == '.'){
                Langkah l = new Langkah(hurufKey, i);
                langkah.add(l);
                i++;
            }
            i = -1;
            while(kolom+i >= 0 && papan[baris][kolom+i] == '.'){
                Langkah l = new Langkah(hurufKey, i);
                langkah.add(l);
```

```
                    i--;
                }
            }else{
                while(baris+panjang+i-1 < A && papan[baris+panjang+i-1][kolom] == '.'){
                    Langkah l = new Langkah(hurufKey, i);
                    langkah.add(l);
                    i++;
                }
                i = -1;
                while(baris+i >= 0 && papan[baris+i][kolom] == '.'){
                    Langkah l = new Langkah(hurufKey, i);
                    langkah.add(l);
                    i--;
                }
            }
        }
        return langkah;
    }

    // Cek berapa mobil yang ada di antara P sampe gerbang
    public int mobilPenghalang(char[][] papan, Map<Character, Map<String, Integer>>
huruf, int tujuanBaris, int tujuanKolom) {
        int penghalang = 0;
        Map<String, Integer> detail = huruf.get('P');
        int barisP = detail.get("baris");
        int kolomP = detail.get("kolom");
        if (huruf.get('P').get("orientasi") == 1) {
            int start = kolomP + detail.get("panjang");
            if (start < tujuanKolom){
                for (int kol = start; kol <= tujuanKolom; kol++) {
                    char blok = papan[barisP][kol];
                    if (blok != '.' && blok != 'P') {
                        penghalang++;
                    }
                }
            }else{
                for (int kol = kolomP - 1; kol >= tujuanKolom; kol--) {
                    char blok = papan[barisP][kol];
                    if (blok != '.' && blok != 'P') {
                        penghalang++;
                    }
                }
            }
        } else {
            int start = barisP + detail.get("panjang");
            if(start < tujuanBaris){
                for (int bar = start; bar <= tujuanBaris; bar++){
                    char blok = papan[bar][kolomP];
                    if (blok != '.' && blok != 'P'){
                        penghalang++;
                    }
                }
```

```
            } else if(start > tujuanBaris){
                for (int bar = start - 1; bar >= tujuanBaris; bar--){
                    char blok = papan[bar][kolomP];
                    if (blok != '.' && blok != 'P'){
                        penghalang++;
                    }
                }
            }
        }
        return penghalang;
    }

    // Cek jarak dari ujung P ke gerbang
    public int jarak(Map<Character, Map<String, Integer>> huruf, int tujuanBaris, int
tujuanKolom){
        if(huruf.get('P').get("orientasi")==1){
            return Math.abs(huruf.get('P').get("kolom") - tujuanKolom + 1);
        }else{
            return Math.abs(huruf.get('P').get("baris") - tujuanBaris + 1);
        }
    }

    // Cek apakah sudah sampai di finish
    public static boolean finish(Map<Character, Map<String, Integer>> huruf, int
tujuanBaris, int tujuanKolom){
        if(huruf.get('P').get("orientasi")==1){
            if(huruf.get('P').get("kolom")< tujuanKolom){
                return huruf.get('P').get("kolom")+huruf.get('P').get("panjang")-1 ==
tujuanKolom;
            }else{
                return huruf.get('P').get("kolom") == tujuanKolom;
            }
        }else{
            if(huruf.get('P').get("baris")< tujuanBaris){
                return huruf.get('P').get("baris")+huruf.get('P').get("panjang")-1 ==
tujuanBaris;
            }else{
                return huruf.get('P').get("baris") == tujuanBaris;
            }
        }
    }
    private static String cetakPapan(char[][] papan, int barisPindah, int kolomPindah,
int panjang, int orientasi) {
        StringBuilder output = new StringBuilder();
        for (int i = 0; i < papan.length; i++) {
            for (int j = 0; j < papan[i].length; j++){
                char karakter = papan[i][j];
                String teks = String.valueOf(karakter);
                if(orientasi == 1){
                    if ((barisPindah == i && j >= kolomPindah && j < kolomPindah +
panjang)) {
                        output.append("\u001B[43m" + getColoredText(teks) +
```

```
"\u001B[0m");
                    } else {
                        output.append(getColoredText(teks));
                    }
                }else{
                    if ((kolomPindah == j && i >= barisPindah && i < barisPindah +
panjang)) {
                        output.append("\u001B[43m" + getColoredText(teks) +
"\u001B[0m");
                    } else {
                        output.append(getColoredText(teks));
                    }
                }
            }
            output.append(System.lineSeparator());
        }
        return output.toString();
    }
    public static String getColoredText(String text) {
        StringBuilder coloredText = new StringBuilder();
        for (char c : text.toCharArray()) {
            if(c == '.'){
                coloredText.append(c);
                continue;
            }
            int index = Character.toUpperCase(c) - 'A';
            coloredText.append(COLORS[index]).append(c).append(RESET);
        }
        return coloredText.toString();
    }
    public static final String RESET = "\u001B[0m";
    public static final String[] COLORS = {
        "\u001B[31m", // A - Merah
        "\u001B[34m", // B - Biru
        "\u001B[32m", // C - Hijau
        "\u001B[33m", // D - Kuning
        "\u001B[35m", // E - Ungu
        "\u001B[36m", // F - Cyan
        "\u001B[91m", // G - Merah terang
        "\u001B[94m", // H - Biru terang
        "\u001B[92m", // I - Hijau terang
        "\u001B[93m", // J - Kuning terang
        "\u001B[95m", // K - Ungu terang
        "\u001B[96m", // L - Cyan terang
        "\u001B[37m", // M - Putih
        "\u001B[90m", // N - Abu-abu
        "\u001B[97m", // O - Putih terang
        "\u001B[30m", // P - Hitam (jika latar belakang terang)
        "\u001B[41m", // Q - Background merah
        "\u001B[42m", // R - Background hijau
        "\u001B[43m", // S - Background kuning
        "\u001B[44m", // T - Background biru
```

```
        "\u001B[45m", // U - Background ungu
        "\u001B[46m", // V - Background cyan
        "\u001B[100m", // W - Background abu-abu gelap
        "\u001B[101m", // X - Background merah terang
        "\u001B[102m", // Y - Background hijau terang
        "\u001B[103m"  // Z - Background kuning terang
    };
}
```

3.2.2 State.java

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class State {
    char[][] board;
    int heuristic;
    Map<Character, Map<String, Integer>> pieces;
    List<Langkah> steps;
    Set<String> visited;

    public State(){
        this.board = new char[0][0];
        this.heuristic = 0;
        this.pieces = new HashMap<>();
        this.steps = new ArrayList<>();
        this.visited = new HashSet<>();
    }
    public State(char[][] board, int heuristic, Map<Character, Map<String, Integer>>
pieces, List<Langkah> steps, Set<String> visited) {
        this.board = board;
        this.heuristic = heuristic;
        this.pieces = pieces;
        this.steps = steps;
        this.visited = visited;
    }
}
```

3.2.3 Langkah.java

```
public class Langkah {
    public Character piece;
    public int jarak;

    public Langkah(Character piece, int jarak) {
```

```
        this.piece = piece;
        this.jarak = jarak;
    }
}
```

Bab 4

4.1 Test Case

Input	Output
UCS 6 6 13 ABCCDD ABEE.F .BPPGFX HHIJGF ..IJKK .LLMM.	Gerakan ke-32: P-kanan 2 CCIJDD EEIJ.. .B..PP ABHHGF ABKKGF .LLMMF Waktu eksekusi: 2046 ms Jumlah state yang dicari: 42621
GBFS (Ada blok yang menghalangi P) 5 5 5 B.... XBAAPP .CCCC .DE.. .DE..	Hasil pencarian GBFS Tidak ditemukan solusi (hapus spasi di akhir tiap baris di file input)

GBFS dengan heuristic banyak mobil

```
4 4
4
   X
.AAA
BCCC
B..P
DD.P
```

```
Gerakan ke-5: P-atas 2
AAAP
CCCP
B...
BDD.


Waktu eksekusi: 3 ms
Jumlah state yang dicari: 26
```

A* dengan heuristik jarak ke pintu

```
14 6
12
.A....
.A....
BBB..C
.....C
...P.D
EEEP.D
......
.F....
.FGGG.
.F...H
III..H
JJJ..K
.....K
LLL...
   X
```

```
Gerakan ke-3: P-bawah 8
.A....
.A....
BBB..C
.....C
.....D
EEE..D
.F....
.F....
GGG...
.F...H
III..H
JJJ..K
...P.K
LLLP..


Waktu eksekusi: 184 ms
Jumlah state yang dicari: 7395
```

Lampiran

Pranala Repository

https://github.com/qodriazka/Tucil3_13523010

Tabel Ketercapaian

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	V	
2. Program berhasil dijalankan	V	
3. Solusi yang diberikan program benar dan mematuhi aturan permainan	V	
4. Program dapat membaca masukan berkas .txt dan menyimpan solusi berupa print board tahap per tahap dalam berkas .txt	V	
5. [Bonus] Implementasi algoritma pathfinding alternatif		V
6. [Bonus] Implementasi 2 atau lebih heuristik alternatif	V	
7. [Bonus] Program memiliki GUI		V
8. Program dan laporan dibuat (kelompok) sendiri	V	