

Laporan Tugas Kecil 3
IF2211 Strategi Algoritma

Pemanfaatan Algoritma-Algoritma *Pathfinding* Dalam Penyelesaian Puzzle

Rush Hour

Disusun oleh:

Dzubyan Ilman Ramadhan (10122010)



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

2025

Algoritma-Algoritma Pathfinding

Apabila BFS & DFS mencari simpul-simpul berdasarkan konfigurasi graf, kita juga dapat melakukan pencarian lintasan yang optimal dari suatu graf dari suatu simpul ke tujuan. Ini dilakukan dengan cara menghitung ongkos yang dibutuhkan untuk mencapai ke suatu tujuan atau taksiran ongkos untuk mencapai tujuan yang disebut heuristik dan memprioritaskan simpul yang ongkosnya paling minimal. Terdapat 3 tipe algoritma yang menggunakan konsep ini yaitu : Greedy Best First Search, Uniform Cost Search, dan A-Star (A*)

Greedy Best First Search (GBFS)

Pada GBFS, ongkos yang dilihat hanyalah taksiran ongkos untuk mencapai target tanpa melihat ongkos dari simpul awal ke simpul saat itu.

Garis besar algoritmanya adalah :

1. Dari simpul kondisi awal, cari semua tetangga dari simpul tersebut yang belum dijelajahi
2. Hitung taksiran heuristik untuk tiap tetangga
3. Simpan dalam Queue Prioritas berdasarkan nilai $h(n)$
4. Ambil simpul paling depan dalam queue dan ulangi step 1.
5. Apabila simpul adalah target, pencarian selesai
6. Apabila Queue menjadi kosong, solusi tidak ada.

Uniform Cost Search (UCS)

Pada UCS, ongkos yang dilihat hanyalah taksiran ongkos saat itu dari asal ke simpul saat itu. Metode ini tidak menggunakan heuristik taksiran ongkos ke simpul tujuan.

Garis besar algoritmanya adalah :

1. Dari simpul kondisi awal, cari semua tetangga dari simpul tersebut yang belum dijelajahi
2. Hitung jarak simpul dari simpul awal untuk tiap tetangga
3. Simpan dalam Queue Prioritas berdasarkan nilai jarak
4. Ambil simpul paling depan dalam queue dan ulangi step 1.
5. Apabila simpul adalah target, pencarian selesai
6. Apabila Queue menjadi kosong, solusi tidak ada.

A-Star (A*)

Pada metode ini, kita menghitung ongkos suatu kandidat solusi dengan gabungan dari ongkos dari simpul asal ke simpul saat itu dan juga heuristik taksiran ongkos dari simpul saat itu dan tujuan. A-Star akan memperoleh solusi optimal serta kompleksitas waktunya akan jauh lebih rendah daripada UCS. Heuristik yang dipakai bisa merupakan fungsi mana saja asalkan fungsi tidak pernah lebih dari ongkos sebenarnya ke target

1. Dari simpul kondisi awal, cari semua tetangga dari simpul tersebut yang belum dijelajahi
2. Hitung fungsi ongkos untuk tiap tetangga yang dihasilkan, yaitu jarak simpul saat ini dari simpul awal dan heuristik pada simpul tersebut
3. Simpan dalam Queue Prioritas berdasarkan nilai fungsi ongkos
4. Ambil simpul paling depan dalam queue dan ulangi step 1.
5. Apabila simpul adalah target, pencarian selesai
6. Apabila Queue menjadi kosong, solusi tidak ada.

Analisis Algoritma

Dalam algoritma pathfinding, kita perlu menentukan fungsi yang baik sebagai penentuan prioritas dalam eksplorasi simpul. Umumnya, fungsi tersebut dapat dituliskan sebagai

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

dengan $g(n)$ adalah fungsi yang menyatakan ongkos dari simpul awal ke simpul n , dan $h(n)$, biasa disebut fungsi heuristik, adalah fungsi yang menyatakan taksiran ongkos dari simpul n ke tujuan

Dalam permainan Rush Hour dengan objektif menggerakkan mobil-mobil lain sehingga mobil utama yang berwarna merah bisa keluar, fungsi $h(n)$ yang cocok digunakan adalah sebagai berikut: hitung banyak mobil yang berada di antara mobil P dan pintu keluar. Lalu, fungsi $g(n)$ yang cocok adalah jarak terpendek simpul n dari simpul awal.

Fungsi heuristik dapat dipastikan admissible karena apabila terdapat k mobil di antara mobil merah dan pintu keluar, kita perlu minimal k gerakan untuk memindahkan mobil tersebut serta memindahkan P ke pintu keluar.

Karena tiap langkah memiliki ongkos 1, maka UCS pada kasus ini akan sama saja dengan melakukan BFS pada pencarian ruang status dari permainan ini.

A^* secara teoritis akan lebih efisien dalam pencarian solusi daripada UCS karena UCS harus mengeksplor semua kemungkinan dengan sama, namun A^* memprioritaskan langkah yang mendekatkan kita dengan solusi. Jadi, solusi optimal akan diperoleh lebih cepat.

Tak seperti A^* dan UCS, pencarian GBFS tidak selalu membuahkan solusi optimal. Ini karena GBFS tidak memperhatikan taksiran ongkos dari suatu simpul ke simpul awal yang bisa saja jauh lebih besar, dan hanya memperdulikan perolehan solusi.

Source Code

App.java

```
package dvp;
import dvp.utils.*;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Set;
import java.util.Scanner;
import java.io.BufferedReader;

/**
 * Hello world!
 *
 */
public class App
{
    int A = 0, B = 0, N = 0;
    int[] exit_location = {0, 0};
    int[] dimension = {0, 0};
    Board board;
    ArrayList<Piece> gamePiece = new ArrayList<Piece>();
    String method;
    int nodeCount;

    public static ArrayList<String> readAllLines(String filePath) {
        ArrayList<String> lines = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                lines.add(line);
            }
        }
    }
}
```

```

        System.out.println(String.format("File input di %s
terbaca", filePath));
    } catch (IOException e) {
        System.err.println("Error membaca file: " +
e.getMessage());
    }

    return lines;
}

public static int[] getBoardSizeInput(String line) {
    int[] boardSizeInput = new int[2];
    try {
        // Split the line by whitespace or comma (adjust delimiter
as needed)
        String[] parts = line.trim().split(" ");

        if (parts.length == 2) {
            boardSizeInput[0] = Integer.parseInt(parts[0]);
            boardSizeInput[1] = Integer.parseInt(parts[1]);
        } else {
            throw new IllegalArgumentException("Ada kesalahan di
formatting input");
        }
    } catch (NumberFormatException e) {
        System.err.println("Error membaca input di baris : " +
line);
    }
    return boardSizeInput;
}

public static int getNumOfPieces(String line) {
    int N = 0;
    try {
        N = Integer.parseInt(line.trim());
    } catch (NumberFormatException e) {
        System.err.println("Error membaca input di baris : " +
line);
    }

    return N;
}

```

```

public void getPieces(int A, int B, int N, ArrayList<String> lines)
{
    int lines_row = lines.size();
    int lines_col = lines.get(0).length();
    if(!((lines_row == A && lines_col == B + 1) || (lines_row == A
+ 1 && lines_col == B))) {
        throw new IllegalArgumentException("Input di txt tidak
sesuai");
    } else if ((lines_row == A + 1 && lines_col == B)) {
        String firstLine = lines.get(0);
        String lastLine = lines.get(lines.size() - 1);
        if (firstLine.trim().equals("K")) {
            exit_location[0] = 1;
            exit_location[1] = firstLine.indexOf("K") ;
            lines.remove(0);
        } else if (lastLine.trim().equals("K")) {
            exit_location[0] = 3;
            exit_location[1] = lastLine.indexOf("K");
            lines.remove(lines.size() - 1);
        }

        Set<String> nameTags = new HashSet<String>();
        String letter = "";
        for(int i = 0; i < A; i++) {
            for(int j = 0; j < B; j++) {
                letter =
Character.toString(lines.get(i).charAt(j));

                if(letter.equals(".")) {
                    continue;
                } else if(!nameTags.contains(letter)) {
                    nameTags.add(letter);
                    Piece piece = new Piece(letter, i, j);
                    gamePiece.add(piece);
                } else if(nameTags.contains(letter)) {
                    for(Piece piece : gamePiece) {
                        if(piece.getPieceName().equals(letter)) {
                            piece.addPosition(i, j);
                        }
                    }
                }
            }
        }
    }
}

```

```

    } else if (lines_row == A && lines_col == B + 1) {
        Set<String> nameTags = new HashSet<String>();
        String letter = "";
        String firstLetter =
Character.toString(lines.get(0).charAt(0));
        String lastLetter =
Character.toString(lines.get(0).charAt(B));

        if(firstLetter.equals(" ") || firstLetter.equals("K")) {
            for(int i = 0; i < A; i++) {
                for(int j = -1; j < B; j++) {
                    letter =
Character.toString(lines.get(i).charAt(j+1));
                    if (letter.equals(" ") || letter.equals(".")) {
                        continue;
                    } else if (letter.equals("K") && j == -1) {
                        exit_location[0] = 2;
                        exit_location[1] = i;
                    } else if (!nameTags.contains(letter)) {
                        nameTags.add(letter);
                        Piece piece = new Piece(letter, i, j);
                        gamePiece.add(piece);
                    } else if (nameTags.contains(letter)) {
                        for(Piece piece : gamePiece) {
                            if(piece.getPieceName().equals(letter))
{
                                piece.addPosition(i, j);
                            }
                        }
                    }
                }
            }
        } else if (lastLetter.equals(" ") ||
lastLetter.equals("K")) {
            for(int i = 0; i < A; i++) {
                for(int j = 0; j <= B; j++) {
                    letter =
Character.toString(lines.get(i).charAt(j));
                    if (letter.equals(" ") || letter.equals(".")) {
                        continue;
                    } else if (letter.equals("K") && j == B) {

```



```

        closedSet.add(boardStr);

        for(Move move : current.getState().generateSuccessor()) {
            String nextBoardStr =
move.getResultState().displayBoard();
            if (!closedSet.contains(nextBoardStr)) {
                SearchNode nextNode = new
SearchNode(move.getResultState(), current, move.getMoveDesc());
                openSet.add(nextNode);
            }
        }
    }

    return new ArrayList<>();
}

public static List<SearchNode> reconstructPath(SearchNode goalNode)
{
    List<SearchNode> path = new ArrayList<>();
    SearchNode current = goalNode;

    while (current != null) {
        path.add(current);
        current = current.getParent();
    }

    Collections.reverse(path);
    return path;
}

public static String printSolution(List<SearchNode> solution,
boolean isWithColor) {
    StringBuilder sb = new StringBuilder("");
    if (solution.isEmpty()) {
        sb.append("Tidak ada solusi yang ditemukan!");
    } else {
        sb.append("Solusi ditemukan pada " + (solution.size() - 1) +
" gerakan: \n");
        for (int i = 1; i < solution.size(); i++) {
            SearchNode node = solution.get(i);
            sb.append("Gerakan " + i + ": " +
node.getMoveDesc() + "\n");

```

```

        String piece_name =
Character.toString(node.getMoveDesc().charAt(0));
        if(isWithColor) {

sb.append(node.getState().displayBoard(piece_name));
        } else {

sb.append(node.getState().displayBoardNoColor());
        sb.append("\n");
        }
    }
}

return sb.toString();
}

public static void main( String[] args ) {
    System.out.println("Masukkan nama file txt yang dijadikan input
(pakai .txt di akhir)");
    Scanner scanner = new Scanner(System.in);
    String inputName = scanner.nextLine();
    String filePath = "test\\" + inputName;

    App game = new App();
    ArrayList<String> lines = readAllLines(filePath);
    int[] dimension = getBoardSizeInput(lines.get(0));
    game.dimension = dimension;
    game.A = dimension[0];
    game.B = dimension[1];

    game.N = getNumOfPieces(lines.get(1));
    game.getPieces(game.A, game.B, game.N, new
ArrayList<>(lines.subList(2, lines.size())));
    for(Piece p : game.gamePiece) {
        System.out.println(p.getPieceName() + " " + p.getRow() + "
" + p.getCol());
    }
    game.board = new Board(game.A, game.B, game.gamePiece,
game.exit_location);

    System.out.println("Pilih algoritma yang ingit digunakan");
    System.out.println("Greedy Best First Search (G) | USC (U) |
A-Star (A)");
}

```

```

        game.method = scanner.nextLine();
        scanner.close();

        System.out.println("Initial state:");
        System.out.println(game.board.displayBoard());

        long startTime = System.currentTimeMillis();
        List<SearchNode> solution = game.solve(game.board,
game.method);
        long endTime = System.currentTimeMillis();

        long timeElapsed = endTime - startTime;
        System.out.println(printSolution(solution, true));
        System.out.println("Waktu yang dibutuhkan : " + timeElapsed + "
ms\n");

        System.out.println("Banyak simpul yang dikunjungi : " +
game.nodeCount);

        try {
            FileWriter writer = new FileWriter("test\\output.txt");
            writer.write(printSolution(solution, false) + "\n");
            writer.write("Waktu yang dibutuhkan : " + timeElapsed + "
ms\n");

            writer.write("Banyak simpul yang dikunjungi : " +
game.nodeCount);
            writer.close(); // Always close the writer
            System.out.println("Solusi ditulis ke solution.txt di
folder test");
        } catch (IOException e) {
            System.out.println("Ada error menulis solusi ke file
text.");
            e.printStackTrace();
        }
    }
}

```

Board.java

```

package dvp.utils;

import java.util.ArrayList;
import java.util.Comparator;

```

```

import java.util.Objects;

public class Board {
    public static final String RESET = "\u001B[0m";
    public static final String RED = "\u001B[31m";
    public static final String GREEN = "\u001B[32m";
    public static final String BLUE = "\u001B[34m";

    private final int row_size;
    private final int column_size;
    private Piece[][] grid = null;
    private int[] exit_location; // Koordinat 0 = posisinya, koordinat
1 == angka barisannya. 1 == Atas, mutar lawan jarum jam
    private Piece main_car;
    private int main_car_idx;
    private ArrayList<Piece> gamePieces;

    public Board(int row_size, int column_size, ArrayList<Piece>
gamePieces, int[] exit_location) {
        if (row_size <= 0 || column_size <= 0) {
            throw new IllegalArgumentException("Ukuran papan tidak
mungkin negatif atau nol");
        }

        this.row_size = row_size;
        this.column_size = column_size;
        this.grid = new Piece[row_size][column_size];

        findAndGetPCar(gamePieces);

        if(!isExitAligned(gamePieces, exit_location)) {
            throw new IllegalArgumentException("Puzzle tidak mungkin
diselesaikan karena mobil P tidak bisa keluar");
        }

        this.exit_location = exit_location;

        if(!isCarInFrontP(gamePieces)) {
            throw new IllegalArgumentException("Ada mobil yang di depan
mobil merah (P)");
        }

        this.gamePieces = gamePieces;

```

```

    }

    public Board(Board other) {
        this.column_size = other.column_size;
        this.row_size = other.row_size;
        this.exit_location = other.exit_location;
        this.gamePieces = new ArrayList<>();
        for (Piece p : other.gamePieces) {
            this.gamePieces.add(new Piece(p));
            if(p.getPieceName().equals("P")) {
                this.main_car = new Piece(p);
            }
        }

        this.main_car_idx = other.main_car_idx;
        this.grid = other.grid;
    }

    public String displayBoard() {
        StringBuilder sb = new StringBuilder();
        placePieces();
        if(exit_location[0] == 1) {
            String gate = " ".repeat(exit_location[1]) + GREEN + "K" +
RESET + " ".repeat(column_size - exit_location[1] + 1);
            sb.append(gate + "\n");
        }

        for (int i = 0; i < row_size; i++) {
            for(int j = 0; j < column_size; j++) {
                if(j == 0 && exit_location[0] == 2 && i ==
exit_location[1]) {
                    sb.append(GREEN + "K" + RESET);
                } else if (j == 0 && exit_location[0] == 2) {
                    sb.append(" ");
                }

                if (grid[i][j] == null) {
                    sb.append(".");
                } else if(grid[i][j].getPieceName().equals("P")) {
                    sb.append(RED + "P" + RESET);
                } else {
                    sb.append(grid[i][j].getPieceName());
                }
            }
        }
    }

```

```

        if(j == column_size - 1 && exit_location[0] == 4 && i
== exit_location[1]) {
            sb.append(GREEN + "K" + RESET);
        } else if (j == column_size - 1 && exit_location[0] ==
4) {
            sb.append(" ");
        }
    }

    sb.append("\n");
}

if(exit_location[0] == 3) {
    String gate = " ".repeat(exit_location[1]) + GREEN + "K" +
RESET + " ".repeat(column_size - exit_location[1] + 1);
    sb.append(gate);
}

clearBoard();
return sb.toString();
}

public String displayBoardNoColor() {
    StringBuilder sb = new StringBuilder();
    placePieces();
    if(exit_location[0] == 1) {
        String gate = " ".repeat(exit_location[1]) + "K" + "
".repeat(column_size - exit_location[1] + 1);
        sb.append(gate + "\n");
    }

    for (int i = 0; i < row_size; i++) {
        for(int j = 0; j < column_size; j++) {
            if(j == 0 && exit_location[0] == 2 && i ==
exit_location[1]) {
                sb.append("K");
            } else if (j == 0 && exit_location[0] == 2) {
                sb.append(" ");
            }

            if (grid[i][j] == null) {
                sb.append(".");
            }
        }
    }
}

```

```

        } else if(grid[i][j].getPieceName().equals("P")) {
            sb.append("P");
        } else {
            sb.append(grid[i][j].getPieceName());
        }

        if(j == column_size - 1 && exit_location[0] == 4 && i
== exit_location[1]) {
            sb.append("K");
        } else if (j == column_size - 1 && exit_location[0] ==
4) {
            sb.append(" ");
        }
    }

    sb.append("\n");
}

if(exit_location[0] == 3) {
    String gate = " ".repeat(exit_location[1]) + "K" + "
".repeat(column_size - exit_location[1] + 1);
    sb.append(gate);
}

clearBoard();
return sb.toString();
}

public String displayBoard(String piece_name) {
    StringBuilder sb = new StringBuilder();
    placePieces();
    if(exit_location[0] == 1) {
        String gate = " ".repeat(exit_location[1]) + GREEN + "K" +
RESET + " ".repeat(column_size - exit_location[1] + 1);
        sb.append(gate + "\n");
    }

    for (int i = 0; i < row_size; i++) {
        for(int j = 0; j < column_size; j++) {
            if(j == 0 && exit_location[0] == 2 && i ==
exit_location[1]) {
                sb.append(GREEN + "K" + RESET);
            } else if (j == 0 && exit_location[0] == 2) {

```

```

        sb.append(" ");
    }

    if (grid[i][j] == null) {
        sb.append(".");
    } else if (grid[i][j].getPieceName().equals("P")) {
        sb.append(RED + "P" + RESET);
    } else if
(grid[i][j].getPieceName().equals(piece_name)) {
        sb.append(BLUE + grid[i][j].getPieceName() +
RESET);

    } else {
        sb.append(grid[i][j].getPieceName());
    }

    if (j == column_size - 1 && exit_location[0] == 4 && i
== exit_location[1]) {
        sb.append(GREEN + "K" + RESET);
    } else if (j == column_size - 1 && exit_location[0] ==
4) {

        sb.append(" ");
    }

    sb.append("\n");
}

if (exit_location[0] == 3) {
    String gate = " ".repeat(exit_location[1]) + GREEN + "K" +
RESET + " ".repeat(column_size - exit_location[1] + 1);
    sb.append(gate + "\n");
}

clearBoard();
return sb.toString();
}

public void placePieces() {
    for (Piece piece : gamePieces) {
        int anchor_row = piece.getRow();
        int anchor_col = piece.getCol();

```



```

        if(anchor_row >= row_size || anchor_row < 0 || anchor_col
>= column_size || anchor_col < 0) {
            throw new IllegalArgumentException("Ada piece di luar
papan");
        }

        if(piece.getisVertical() && (anchor_row + piece.getSize() >
row_size)) {
            throw new IllegalArgumentException("Vertical piece
extends off the board");
        }
        if(!piece.getisVertical() && (anchor_col + piece.getSize()
> column_size)) {
            throw new IllegalArgumentException("Horizontal piece
extends off the board");
        }

        if(piece.getisVertical()) {
            for (int i = 0; i < piece.getSize(); i++) {
                if(grid[anchor_row + i][anchor_col] != null) {
                    throw new IllegalStateException("Position sudah
diisi");
                }
                grid[anchor_row + i][anchor_col] = piece;
            }
        } else {
            for (int i = 0; i < piece.getSize(); i++) {
                if(grid[anchor_row][anchor_col + i] != null) {
                    throw new IllegalStateException("Posisi sudah
diisi");
                }
                grid[anchor_row][anchor_col + i] = piece;
            }
        }
    }
}

public boolean isWinState() {
    boolean isWin = false;
    Piece main_car = gamePieces.get(main_car_idx);
    switch (exit_location[0]) {
        case 1:
            if (main_car.getRow() == 0) {

```

```

        isWin = true;
    }
    break;
    case 2:
        if(main_car.getCol() == 0) {
            isWin = true;
        }
        break;
    case 3:
        if(main_car.getRow() + main_car.getSize() - 1 == row_size -
1) {
            isWin = true;
        }
        break;
    case 4:
        if(main_car.getCol() + main_car.getSize() - 1 ==
column_size - 1) {
            isWin = true;
        }
    }

    return isWin;
}

public ArrayList<Move> generateSuccessor() {
    ArrayList<Move> possibleMoves = new ArrayList<>();
    placePieces();
    for(int i = 0; i < gamePieces.size(); i++) {
        Piece p = gamePieces.get(i);
        int anchor_row = p.getRow();
        int anchor_col = p.getCol();
        int size = p.getSize();
        if(p.getisVertical()) {
            if(anchor_row > 0 && grid[anchor_row - 1][anchor_col]
== null) {
                Board newState = new Board(this);
                Piece newPiece = newState.gamePieces.get(i);
                newPiece.setRow(newPiece.getRow() - 1);
                possibleMoves.add(new Move(newState, i,
newPiece.getPieceName(), Direction.Atas));
            } if(anchor_row + size < row_size && grid[anchor_row +
size][anchor_col] == null) {
                Board newState = new Board(this);

```

```

        Piece newPiece = newState.gamePieces.get(i);
        newPiece.setRow(newPiece.getRow() + 1);
        possibleMoves.add(new Move(newState, i,
newPiece.getPieceName(), Direction.Bawah));
    }
    } else {
        if(anchor_col > 0 && grid[anchor_row][anchor_col - 1]
== null) {
            Board newState = new Board(this);
            Piece newPiece = newState.gamePieces.get(i);
            newPiece.setCol(newPiece.getCol() - 1);
            possibleMoves.add(new Move(newState, i,
newPiece.getPieceName(), Direction.Kiri));
        } if(anchor_col + size < column_size &&
grid[anchor_row][anchor_col + size] == null) {
            Board newState = new Board(this);
            Piece newPiece = newState.gamePieces.get(i);
            newPiece.setCol(newPiece.getCol() + 1);
            possibleMoves.add(new Move(newState, i,
newPiece.getPieceName(), Direction.Kanan));
        }
    }
}

clearBoard();
return possibleMoves;
}

public String getStateHash() {
    StringBuilder sb = new StringBuilder("");
    ArrayList<Piece> sortedPieces = new ArrayList<>(gamePieces);
    sortedPieces.sort(Comparator.comparing(Piece::getPieceName));

    for (Piece p : sortedPieces) {
        sb.append(p.getPieceName()).append(":");
        sb.append(p.getRow()).append(",");
        sb.append(p.getCol()).append(";");
    }

    return sb.toString();
}

@Override

```

```

public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Board state = (Board) o;

    if (gamePieces.size() != state.gamePieces.size()) return false;

    for (int i = 0; i < gamePieces.size(); i++) {
        if (!gamePieces.get(i).equals(state.gamePieces.get(i))) {
            return false;
        }
    }

    return true;
}

@Override
public int hashCode() {
    return Objects.hash(gamePieces);
}

public void findAndGetPCar(ArrayList<Piece> gamePieces) {
    for(int i = 0; i < gamePieces.size(); i++) {
        if(gamePieces.get(i).getPieceName().equals("P")) {
            main_car = gamePieces.get(i);
            main_car_idx = i;
            return;
        }
    }

    throw new IllegalArgumentException("Tidak ada mobil berlabel
P");
}

public void clearBoard() {
    for (int i = 0; i < row_size; i++) {
        for(int j = 0; j < column_size; j++) {
            grid[i][j] = null;
        }
    }
}

public boolean isCarInFrontP(ArrayList<Piece> gamePieces) {

```

```

        for (Piece piece : gamePieces) {
            if(piece.getPieceName().equals("P")) {
                continue;
            }

            if(main_car.getisVertical() && piece.getisVertical() &&
(piece.getCol() == main_car.getCol())) {
                if(exit_location[0] == 1 && piece.getRow() <
main_car.getRow()) {
                    return false;
                } else if (exit_location[0] == 3 && piece.getRow() >
main_car.getRow()) {
                    return false;
                }
            } else if(!main_car.getisVertical() &&
!piece.getisVertical() && (piece.getRow() == main_car.getRow())) {
                if(exit_location[0] == 2 && piece.getCol() <
main_car.getCol()) {
                    return false;
                } else if (exit_location[0] == 4 && piece.getCol() >
main_car.getCol()) {
                    return false;
                }
            }
        }

        return true;
    }

    public boolean isExitAligned(ArrayList<Piece> gamePieces, int[]
exit_location) {
        if ((exit_location[0] == 1 || exit_location[0] == 3) &&
main_car.getCol() == exit_location[1] && main_car.getisVertical()) {
            return true;
        } else if ((exit_location[0] == 2 || exit_location[0] == 4) &&
main_car.getRow() == exit_location[1] && !main_car.getisVertical()) {
            return true;
        }

        return false;
    }

    public int getRowSize() {

```

```

        return row_size;
    }

    public int getColSize() {
        return column_size;
    }

    public int getExitLocationOrientation() {
        return exit_location[0];
    }

    public int getExitLocationPosition() {
        return exit_location[1];
    }

    public Piece getMainPiece() {
        return main_car;
    }

    public int getMainCarIdx() {
        return main_car_idx;
    }

    public ArrayList<Piece> getGamePieces() {
        return gamePieces;
    }

    private Piece[][] deepCopyGrid(Piece[][] original) {
        if (original == null) return null;
        Piece[][] copy = new Piece[original.length][];
        for (int i = 0; i < original.length; i++) {
            copy[i] = original[i].clone();
        }
        return copy;
    }

    public Piece[][] getGridConfig() {
        placePieces();
        Piece[][] result = deepCopyGrid(grid);
        clearBoard();
        return result;
    }
}

```

```
}
```

SearchNode.java

```
package dvp.utils;

import java.util.Objects;

public class SearchNode implements Comparable<SearchNode>{
    private Board state;
    private SearchNode parent;
    private String moveDescription;
    private int gScore;
    private int fScore;
    private String method;

    public SearchNode(Board state, String method) {
        this.state = state;
        this.parent = null;
        this.moveDescription = "Initial state";
        this.method = method;
        if(method.equals("G")) {
            this.gScore = 0;
            this.fScore = calculateHeuristic();
        } else if(method.equals("U")) {
            this.gScore = 0;
            this.fScore = 0;
        } else if(method.equals("A")) {
            this.gScore = 0;
            this.fScore = calculateHeuristic();
        } else {
            throw new IllegalArgumentException("Invalid method");
        }
    }

    public SearchNode(Board state, SearchNode parent, String
moveDescription) {
        this.state = state;
        this.parent = parent;
        this.moveDescription = moveDescription;
        this.method = parent.method;

        if(method.equals("G")) {
```

```

        this.gScore = 0;
        this.fScore = calculateHeuristic();
    } else if(method.equals("U")) {
        this.gScore = parent.gScore + 1;
        this.fScore = parent.gScore + 1;
    } else if(method.equals("A")) {
        this.gScore = parent.gScore + 1;
        this.fScore = this.gScore + calculateHeuristic();
    } else {
        throw new IllegalArgumentException("Invalid method");
    }
}

public int calculateHeuristic() {
    Piece main_piece = state.getMainPiece();

    int distanceToExit = 0;
    int posAfterPiece = 0;
    int blockingVehicles = 0;
    Piece[][] grid = state.getGridConfig();

    switch (state.getExitLocationOrientation()) {
        //Asumsi pas sudah menang, tidak dicek lagi heuristiknya
        case 1:
            distanceToExit = main_piece.getRow();
            posAfterPiece = main_piece.getRow() - 1;

            for (int c = posAfterPiece; c >= 0; c--) {
                if (grid[c][main_piece.getCol()] != null) {
                    blockingVehicles++;
                }
            }
            break;
        case 2:
            distanceToExit = main_piece.getCol();
            posAfterPiece = main_piece.getCol() - 1;

            // Check each position between the target vehicle and the
exit
            for (int c = posAfterPiece; c >= 0; c--) {
                if (grid[main_piece.getRow()][c] != null) {
                    blockingVehicles++;
                }
            }
        }
    }
}

```



```

        }
        break;
        case 3:
            distanceToExit = state.getRowSize() - (main_piece.getRow()
+ main_piece.getSize());
            posAfterPiece = main_piece.getRow() + main_piece.getSize();
            for (int c = posAfterPiece; c < state.getRowSize(); c++) {
                if (grid[c][main_piece.getCol()] != null) {
                    blockingVehicles++;
                }
            }
            break;
        case 4:
            distanceToExit = state.getColSize() - (main_piece.getCol()
+ main_piece.getSize());
            posAfterPiece = main_piece.getCol() + main_piece.getSize();
            for (int c = posAfterPiece; c < state.getColSize(); c++) {
                if (grid[main_piece.getRow()][c] != null) {
                    blockingVehicles++;
                }
            }
            break;
    }

    return distanceToExit + blockingVehicles;
}

@Override
public int compareTo(SearchNode other) {
    return Integer.compare(this.fScore, other.fScore);
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    SearchNode that = (SearchNode) o;
    return state.equals(that.state);
}

@Override
public int hashCode() {

```

```

        return Objects.hash(state);
    }

    public Board getState() {
        return state;
    }

    public SearchNode getParent() {
        return parent;
    }

    public String getMoveDesc() {
        return moveDescription;
    }

    public int getGScore() {
        return gScore;
    }

    public int getFScore() {
        return fScore;
    }
}

```

Move.java

```

package dvp.utils;

public class Move {
    Board resultState;
    String piece_name;
    Direction dir;
    int piece_id;
    String moveDescription;

    public Move(Board resultState, int piece_id, String piece_name,
Direction dir) {
        this.resultState = resultState;
        this.piece_id = piece_id;
        this.piece_name = piece_name;
        this.dir = dir;
        this.moveDescription = piece_name + "-" + dir.toString();
    }
}

```

```

    }

    public Board getResultState() {
        return resultState;
    }

    public String getPieceName() {
        return piece_name;
    }

    public Direction getDir() {
        return dir;
    }

    public String getMoveDesc() {
        return moveDescription;
    }

    public int getPieceId() {
        return piece_id;
    }
}

```

Direction.java

```

package dvp.utils;

public enum Direction {
    Atas,
    Bawah,
    Kiri,
    Kanan
}

```

Piece.java

```

package dvp.utils;

import java.util.Objects;

public class Piece {

```

```

private final String piece_name;
private int height = 1;
private int width = 1;
private int size = 1;
private Boolean isVertical = null;
private int row;
private int col;

public Piece(String piece_name, int row, int col) {
    final String validName_String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    if(!validName_String.contains(String.valueOf(piece_name))) {
        throw new IllegalArgumentException("K tidak boleh digunakan
untuk blok");
    }

    this.piece_name = piece_name;
    this.row = row;
    this.col = col;
}

public Piece(Piece other) {
    this.piece_name = other.piece_name;
    this.row = other.row;
    this.col = other.col;
    this.height = other.height;
    this.width = other.width;
    this.size = other.size;
    this.isVertical = other.isVertical;
}

public void addPosition(int i, int j) {
    if (i == row) {
        isVertical = false;
        incWidth();
        if (width > 3) {
            throw new IllegalArgumentException("Ada blok yang
tingginya lebih dari 3");
        }
    } else if (j == col) {
        isVertical = true;
        incHeight();
        if (height > 3) {

```

```

        throw new IllegalArgumentException("Ada blok yang
lebar nya lebih dari 3");
    }
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Piece piece = (Piece) o;
    return row == piece.row &&
        col == piece.col &&
        size == piece.size &&
        isVertical == piece.isVertical;
}

@Override
public int hashCode() {
    return Objects.hash(row, col, size, isVertical);
}

public int getHeight() {
    return height;
}

public int getWidth() {
    return width;
}

public int getSize() {
    return size;
}

public String getPieceName() {
    return piece_name;
}

public boolean getisVertical() {
    return isVertical;
}

public int getRow() {

```

```
        return row;
    }

    public int getCol() {
        return col;
    }

    public void setRow(int newRow) {
        row = newRow;
    }

    public void setCol(int newCol) {
        col = newCol;
    }

    public void incHeight() {
        height += 1;
        size += 1;
    }

    public void incWidth() {
        width += 1;
        size += 1;
    }

    public void decHeight() {
        height -= 1;
        size -= 1;
    }

    public void decWidth() {
        width -= 1;
        size -= 1;
    }
}
```

Hasil Tes

Tes 1 (Kasus tidak punya solusi)

Input

```
3 5
3
ABC..
KABCPP
ABC..
```

Output

GBFS :

```
Masukkan nama file txt yang dijadikan input (pakai .txt di akhir)
input2.txt
File input di test\input2.txt terbaca
Pilih algoritma yang ingin digunakan
Greedy Best First Search (G) | USC (U) | A-Star (A)
G
Initial state:
ABC..
KABCPP
ABC..

Tidak ada solusi yang ditemukan!
Waktu yang dibutuhkan : 2 ms

Banyak simpul yang dikunjungi : 1
Solusi ditulis ke solution.txt di folder test
```

USC :

```
Masukkan nama file txt yang dijadikan input (pakai .txt di akhir)
input2.txt
File input di test\input2.txt terbaca
Pilih algoritma yang ingin digunakan
Greedy Best First Search (G) | USC (U) | A-Star (A)
U
Initial state:
  ABC..
  KABCPP
  ABC..

Tidak ada solusi yang ditemukan!
Waktu yang dibutuhkan : 1 ms

Banyak simpul yang dikunjungi : 1
```

A*

```
Masukkan nama file txt yang dijadikan input (pakai .txt di akhir)
input2.txt
File input di test\input2.txt terbaca
Pilih algoritma yang ingin digunakan
Greedy Best First Search (G) | USC (U) | A-Star (A)
A
Initial state:
  ABC..
  KABCPP
  ABC..

Tidak ada solusi yang ditemukan!
Waktu yang dibutuhkan : 1 ms
```


Tes 2 (Sederhana solvable)

Input

4 3

2

K

AA.

BB.

P..

P..

Output

Greedy

Gerakan 2: B-Kanan

K

.AA

.BB

P..

P..

Gerakan 3: P-Atas

K

.AA

PBB

P..

...

Gerakan 4: P-Atas

K

PAA

PBB

...

...

Waktu yang dibutuhkan : 3 ms

Banyak simpul yang dikunjungi : 7

UCS

Gerakan 2: P-Atas

K

AA.

PBB

P..

...

Gerakan 3: A-Kanan

K

.AA

PBB

P..

...

Gerakan 4: P-Atas

K

PAA

PBB

...

...

Waktu yang dibutuhkan : 2 ms

Banyak simpul yang dikunjungi : 9

A*

Gerakan 2: B-Kanan

K

.AA

.BB

P..

P..

Gerakan 3: P-Atas

K

.AA

PBB

P..

...

Gerakan 4: P-Atas

K

PAA

PBB

...

...

Waktu yang dibutuhkan : 0 ms

Banyak simpul yang dikunjungi : 8

Tes 3 (Kesulitan medium)

Input

```
6 6
8
....AA
..BP.C
DDBP.C
E.FFFC
E...GH
E...GH
|
K
```

Output

Greedy

```
EDD.GC
FFFP.G.
..BP.H
..B..H
  K
Gerakan 45: D-Kanan
E.AA.C
E....C
E.DDGC
FFFP.G.
..BP.H
..B..H
  K
Gerakan 46: P-Bawah
E.AA.C
E....C
E.DDGC
FFF.G.
..BP.H
..BP.H
  K

Waktu yang dibutuhkan : 26 ms

Banyak simpul yang dikunjungi : 659
```

UCS

```
EDDP.C
FFFP.C
....GH
....GH
K
Gerakan 9: P-Bawah
E.B.AA
E.B..C
EDD..C
FFFP.C
...PGH
....GH
K
Gerakan 10: P-Bawah
E.B.AA
E.B..C
EDD..C
FFF..C
...PGH
...PGH
K
```

Waktu yang dibutuhkan : 28 ms

Banyak simpul yang dikunjungi : 1179

A*

E.B.AA

E.B..C

EDDP.C

FFFP.C

....GH

....GH

K

Gerakan 9: P-Bawah

E.B.AA

E.B..C

EDD..C

FFFP.C

...PGH

....GH

K

Gerakan 10: P-Bawah

E.B.AA

E.B..C

EDD..C

FFF..C

...PGH

...PGH

K

Waktu yang dibutuhkan : 34 ms

Banyak simpul yang dikunjungi : 634

Tes 4 (Kesulitan Tinggi)

Input

```
6 6
7
AAB..C
..B..C
..BPPCK
D..EFF
DGGE..
...E..
```

Output

Greedy

```
Gerakan 184: F-Kiri
AAB...
..B...
..BPP.K
FF.E.C
DGGE.C
D..E.C
Gerakan 185: B-Bawah
AA....
..B...
..BPP.K
FFBE.C
DGGE.C
D..E.C
Gerakan 186: P-Kanan
AA....
..B...
..B.PPK
FFBE.C
DGGE.C
D..E.C

Waktu yang dibutuhkan : 78 ms

Banyak simpul yang dikunjungi : 2823
```

UCS

Gerakan 74: P-Kanan

D...AA

D.....

..PP.CK

FFBE.C

GGBE.C

..BE..

Gerakan 75: P-Kanan

D...AA

D.....

...PPCK

FFBE.C

GGBE.C

..BE..

Gerakan 76: C-Bawah

D...AA

D.....

...PP.K

FFBE.C

GGBE.C

..BE.C

Gerakan 77: P-Kanan

D...AA

D.....

....PPK

FFBE.C

GGBE.C

..BE.C

Waktu yang dibutuhkan : 123 ms

Banyak simpul yang dikunjungi : 7001

A*

```

Gerakan 75: C-Bawah
D...AA
D.....
...PPCK
FFBE.C
GGBE.C
..BE..
Gerakan 76: C-Bawah
D...AA
D.....
...PP.K
FFBE.C
GGBE.C
..BE.C
Gerakan 77: P-Kanan
D...AA
D.....
....PPK
FFBE.C
GGBE.C
..BE.C

Waktu yang dibutuhkan : 135 ms

Banyak simpul yang dikunjungi : 6551

```

Hasil Analisis Percobaan

Berdasarkan hasil tes dari program, dapat disimpulkan bahwa metode A* adalah metode yang ternyata paling cepat dalam mencari solusi, baik dalam waktu dan juga banyak simpul yang dilihat secara konsisten. Di permainan rush hour dengan p mobil. Paling banyak terdapat 2p cabang untuk tiap langkah. Apabila tiap gerakan dapat dianggap sebagai ongkos 1, maka kompleksitas UCS adalah $O(p^d)$, kompleksitas A* adalah $O(p^d)$ untuk d adalah panjang solusi optimal. Sementara, kompleksitas GBFS adalah $O(p^m)$ dengan m adalah kedalaman maksimum dari pencarian.

Pranala

Kode dapat diakses di : https://github.com/countz-zero/Tucil3_10122010

Lampiran

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	•	
6. [Bonus] Implementasi 2 atau lebih heuristik alternatif	•	
7. [Bonus] Program memiliki GUI	•	
8. Program dan laporan dibuat (kelompok) sendiri	•	