

IF2211 Strategi Algoritma

Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding

Laporan Tugas Kecil

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma
pada Semester IV Tahun Akademik 2024/2025



Dibuat Oleh

Nicholas Andhika Lucas	13523014
Samantha Laqueenna Ginting	13523138

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2025

DAFTAR ISI

DAFTAR ISI	2
BAB 1 LANDASAN TEORI	3
Uninformed & Informed Search	3
Uniform Cost Search (UCS)	3
Greedy Best First Search (GBFS)	5
A* Search	6
BAB 2 ANALISIS ALGORITMA PATHFINDING	8
BAB 3 SOURCE PROGRAM	10
Struktur Repository	10
Main.java	10
Solver.java	13
Board.java	14
State.java	16
Util.java	16
Style.java	27
BAB 4 PENGUJIAN	28
Algoritma Uniform Cost Search (UCS)	28
Algoritma Greedy Best First Search (GBFS)	31
Algoritma A* Search	33
BAB 5 HASIL ANALISIS & KESIMPULAN	37
Uniform Cost Search (UCS)	37
Greedy Best First Search (GBFS)	37
A* Search	37
BAB 6 IMPLEMENTASI BONUS	38
Heuristik 1: Jumlah Kendaraan yang Menghalangi ke Pintu Keluar	38
Heuristik 2: Jarak ke Pintu Keluar	38
Analisis Berdasarkan Pengujian	38
LAMPIRAN	41
A. Github	41
B. Tabel Pemeriksaan	41
REFERENSI	42

BAB 1 LANDASAN TEORI

Uninformed & Informed Search

Searching atau proses pencarian dapat dibagi menjadi dua tipe yaitu Uninformed Search dan Informed Search. Uninformed Search, yang sering disebut Blind Search, menggambarkan bahwa teknik pencarian ini tidak memiliki informasi atau pengetahuan tambahan mengenai kondisi di luar yang telah disediakan oleh definisi masalah. Uninformed Search terdiri atas beberapa algoritma, antara lain

- Breadth First Search
- Depth First Search
- Depth Limited Search
- Iterative Deepening Search
- Uniform Cost Search

Informed Search menggunakan heuristik, yaitu informasi tambahan yang dapat memandu proses pencarian agar lebih efisien. Pencarian ini terdiri atas beberapa algoritma, antara lain

- Greedy Best First Search
- A* Search
- Beam Search
- Hill Climbing
- Iterative Deepening A*

Pada tugas ini, implementasi pemecahan masalah menggunakan algoritma Uniform Cost Search (UCS), Greedy Best First Search (GBFS), dan A* Search.

Uniform Cost Search (UCS)

Uniform Cost Search adalah pencarian BFS yang digeneralisasi. Pencarian ini dilakukan jika seluruh edges pada graf pencarian tidak memiliki cost, atau memiliki biaya yang sama. Jika pada BFS pencarian dilakukan dengan melakukan ekspansi node berdasarkan urutan kedalaman akar, pada UCS, ekspansi dilakukan berdasarkan biaya dari akar. Ekspansi tiap langkah ditentukan berdasarkan biaya terendah atau bisa disebut juga sebagai fungsi $g(n)$, dimana $g(n)$ merupakan jumlah biaya edge dari root menuju node n . Node-node tersebut disimpan menggunakan priority queue.

Berikut adalah algoritma UCS dalam pseudocode,

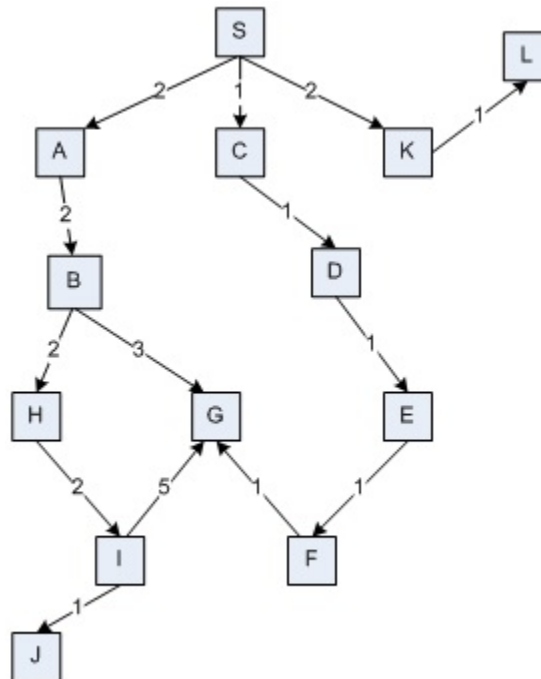
Function $UCS(problem)$ returns a solution, or failure
--

```

node ← a node with State = problem.Initial-State, Path-Cost=0
frontier ← a priority queue ordered by Path-Cost, with node as the only
element
explored ← an empty set
loop do
  if Empty?(frontier) then return failure
  node ← Pop(frontier) /* chooses the lowest-cost node in frontier */
  if problem.Goal-Test(node.State) then return Solution(node)
  add node.State to explored
  for each action in problem.Actions(node.State) do
    child ← Child-Node(problem, node, action)
    if child.State is not in explored or frontier then
      frontier ← Insert(child.frontier)
    else if child.State is in frontier with higher Path-Cost then
      replace that frontier node with child

```

Berikut juga merupakan contoh sederhana dari penyelesaian algoritma UCS menuju node G,



Gambar 1. Contoh Penyelesaian Algoritma UCS

- Frontier atau akar bernilai S.
- Dari S, kita dapat memilih jalur melewati A, C, dan K yang masing-masing jalurnya bernilai 2, 1, dan 2.
- Lakukan ekspansi ke C, yang merupakan jalur paling murah. Dari C, kita dapat melanjutkan ekspansi ke D yang akan bernilai 1. Fungsi $g(n)$ merupakan jumlah cost dari root menuju node n , maka $g(n)$ untuk D dari C adalah $1 + \text{cost sebelumnya menuju C}$, sehingga totalnya adalah 2.
- Seperti langkah sebelumnya, lakukan ekspansi dengan biaya termurah sampai menemukan node G.

Sehingga pemetaan jalur dapat dinyatakan sebagai berikut,

- $f = \{S\}$
- $f = \{C^S, A^S, K^S\} // 1, 2, 2$
- $f = \{A^S, D^C, K^S\} // 2, 2, 2$
- $f = \{D^C, K^S, B^A\} // 2, 2, 4$
- $f = \{K^S, E^D, B^A\} // 2, 3, 4$
- $f = \{E^D, L^K, B^A\} // 3, 3, 4$
- $f = \{L^K, B^A, F^E\} // 3, 4, 4$
- $f = \{B^A, F^E\} // 4, 4$
- $f = \{F^E, H^B, G^B\} // 4, 6, 7$
- $f = \{G^B, H^B\} // 5, 6$
- $f = \{H^B\} // 6$

Jalur UCS = S-C-D-E-F-G

Biaya = 5

Jumlah ekspansi = 10

Greedy Best First Search (GBFS)

Metode pencarian Greedy Best First Search melakukan ekspansi node yang memiliki jarak terdekat dengan targetnya. Namun ekspansi yang dilakukan menggunakan evaluasi node hanya dengan melihat kepada fungsi heuristiknya. Dengan kata lain, yang dibandingkan untuk penentuan ekspansi node adalah nilai estimasi atau prediksinya saja ($f(n) = h(n)$). Algoritma ini cepat, namun tidak optimal dan bisa menghasilkan jalan buntu.

Berikut adalah algoritma GBFS dalam pseudocode,

```
Function GreedyBestFirstSearch(problem) returns a solution, or failure
  node  $\leftarrow$  a node with State = problem.Initial-State
  frontier  $\leftarrow$  a priority queue ordered by Heuristic(node), with node as the
  only element
  explored  $\leftarrow$  an empty set
  loop do
    if Empty?(frontier) then return failure
    node  $\leftarrow$  Pop(frontier) /* chooses the node with lowest heuristic value
  h(n) */
    if problem.Goal-Test(node.State) then return Solution(node)
    add node.State to explored
```

```

for each action in problem.Actions(node.State) do
  child  $\leftarrow$  Child-Node(problem, node, action)
  if child.State is not in explored or frontier then
    frontier  $\leftarrow$  Insert(child, Heuristic(child))
  else if child.State is in frontier with higher Heuristic then
    replace that frontier node with child

```

Menggunakan graf yang sama dengan contoh soal algoritma UCS (lihat Gambar 1), berikut merupakan penyelesaiannya dengan algoritma GBFS,

- Frontier atau akar bernilai S.
- Dari S, kita dapat memilih jalur melewati A, C, dan K yang masing-masing jalurnya bernilai 2, 1, dan 2.
- Lakukan ekspansi ke B, yang merupakan node dengan jarak terdekat ke node G.
- Seperti langkah sebelumnya, lakukan ekspansi dengan jarak terdekat sampai menemukan node G.

Sehingga pemetaan jalur dapat dinyatakan sebagai berikut,

- $f = \{S\}$
- $f = \{A, C, K\} // 2, 4, 5$
- $f = \{B, C, K\} // 3, 4, 5$
- $f = \{G, C, H, K\} // 0, 4, 4, 5$
- $f = \{C, H, K\} // 4, 4, 5$

Jalur GBFS = S-A-B-G

Biaya = 7

Jumlah ekspansi = 4

A* Search

Berbeda dengan GBFS yang hanya melihat kepada nilai $h(n)$, pencarian A* Search juga merupakan bentuk dari Best First Search yang melihat kepada kombinasi nilai dari path-nya yaitu $g(n)$, dengan nilai atau fungsi estimasi yaitu $h(n)$. Sehingga dapat disimpulkan bahwa $f(n) = g(n) + h(n)$.

Berikut adalah algoritma GBFS dalam pseudocode,

```

Function AStarSearch(problem) returns a solution, or failure
  node  $\leftarrow$  a node with State = problem.Initial-State, Path-Cost = 0
  frontier  $\leftarrow$  a priority queue ordered by  $f(n) = g(n) + h(n)$ , with node as
the only element
  explored  $\leftarrow$  an empty set
  loop do
    if Empty?(frontier) then return failure
    node  $\leftarrow$  Pop(frontier) /* chooses the node with lowest  $f(n) = g(n) + h(n)$ 
*/

```

```

if problem.Goal-Test(node.State) then return Solution(node)
add node.State to explored
for each action in problem.Actions(node.State) do
    child  $\leftarrow$  Child-Node(problem, node, action)
    if child.State is not in explored or frontier then
        frontier  $\leftarrow$  Insert(child, child.Path-Cost + Heuristic(child))
    else if child.State is in frontier with higher  $f(n)$  then
        replace that frontier node with child

```

Menggunakan graf yang sama dengan contoh soal algoritma UCS (lihat Gambar 1), berikut merupakan penyelesaiannya dengan algoritma A*,

- Frontier atau akar bernilai S.
- Dari S, kita dapat memilih jalur melewati A, C, dan K yang masing-masing jalurnya bernilai 2, 1, dan 2.
- Lakukan ekspansi ke C, yang merupakan node dengan biaya termurah dan jarak terdekat ke node G.
- Seperti langkah sebelumnya, lakukan ekspansi dengan biaya termurah dan jarak terdekat sampai menemukan node G.

Sehingga pemetaan jalur dapat dinyatakan sebagai berikut,

- $f = \{S\}$
- $f = \{A, C, K\} // 4, 5, 7$
- $f = \{C, K, B\} // 5, 7, 7$
- $f = \{D, K, B\} // 5, 7, 7$
- $f = \{E, K, B\} // 5, 7, 7$
- $f = \{F, K, B\} // 5, 7, 7$
- $f = \{G, K, B\} // 5, 7, 7$
- $f = \{K, B\} // 7, 7$

Jalur GBFS = S-C-D-E-F-G

Biaya = 5

Jumlah ekspansi = 7

BAB 2 ANALISIS ALGORITMA PATHFINDING

Algoritma UCS diimplementasikan dengan memanfaatkan antrian prioritas (PriorityQueue) yang memprioritaskan simpul berdasarkan nilai $g(n)$ saja, yaitu jumlah langkah dari state awal menuju state saat ini. Dalam kelas State, fungsi `getCost()` mencerminkan nilai $g(n)$, dan fungsi `getTotalCost()` akan mengembalikan nilai $f(n)$ yang sesuai dengan algoritma yang sedang digunakan. Jika algoritma yang digunakan adalah UCS, maka `getTotalCost()` hanya mengembalikan nilai `cost`, karena UCS tidak menggunakan heuristik ($h(n) = 0$). Pemilihan state berikutnya untuk diekspansi dilakukan berdasarkan nilai `cost` terkecil, menjadikan UCS menjelajahi semua kemungkinan jalur secara urut berdasarkan total langkah dari awal dan menjamin solusi yang optimal. Dalam kelas Solver, pemilihan algoritma ditentukan oleh parameter `algorithm`, dan `state.getTotalCost(algorithm)` akan mengembalikan nilai $f(n)$ sesuai definisi algoritma yang dipilih.

Meskipun UCS dan BFS memiliki pendekatan yang berbeda, dalam konteks permainan Rush Hour, mereka bisa menunjukkan perilaku yang serupa. Hal ini terjadi karena setiap aksi pada permainan dianggap memiliki `cost` yang sama (misalnya = 1). Dengan asumsi tersebut, UCS akan berperilaku seperti BFS, yaitu menjelajahi node level per level.

Dalam implementasi GBFS (Greedy Best First Search), antrian prioritas memprioritaskan simpul berdasarkan nilai $h(n)$ saja, yaitu estimasi heuristik dari state saat ini ke goal. Di dalam kelas State, jika algoritma yang digunakan adalah Greedy, maka `getTotalCost()` akan mengembalikan nilai dari `Heuristic.evaluate(this)`. Karena nilai $g(n)$ diabaikan, GBFS tidak memperhitungkan jumlah langkah yang telah ditempuh sebelumnya. Akibatnya, GBFS dapat bekerja sangat cepat dalam menemukan solusi yang tampak dekat, namun tidak menjamin solusi tersebut optimal. Dalam permainan Rush Hour, GBFS dapat memilih jalur yang tampak menjanjikan secara heuristik tetapi ternyata memerlukan lebih banyak langkah karena banyaknya kendaraan penghalang yang harus dimanuver.

A* menggabungkan dua komponen evaluasi: $g(n)$ dan $h(n)$ dalam fungsi $f(n) = g(n) + h(n)$. Dalam kelas State, ketika algoritma diset ke A*, maka fungsi `getTotalCost()` akan mengembalikan penjumlahan `getCost()` dan `Heuristic.evaluate(this)`. Hal ini membuat A* mempertimbangkan baik langkah yang telah ditempuh maupun estimasi langkah yang tersisa, menjadikan algoritma ini efisien namun tetap menjamin solusi optimal. File `Solver.java` mengatur jalannya eksekusi algoritma dengan menjaga struktur data priority queue, visited states (melalui Map `visited`), serta backtracking dari goal ke start untuk membentuk solusi akhir.

A* terbukti lebih efisien dibandingkan UCS dalam konteks permainan Rush Hour, karena penggunaan informasi tambahan dari heuristik $h(n)$ membantu memfokuskan eksplorasi ke arah solusi dan

mengurangi jumlah node yang perlu diekspansi. UCS yang tidak menggunakan $h(n)$ justru cenderung mengeksplorasi lebih banyak simpul.

Secara umum, dalam algoritma pathfinding, fungsi evaluasi $f(n)$ digunakan untuk menentukan prioritas ekspansi simpul di dalam antrian prioritas. Fungsi ini berperan penting dalam menentukan urutan eksplorasi simpul dan dirumuskan sebagai berikut:

- UCS:
 - $g(n)$ = jumlah langkah dari awal sampai simpul saat ini
 - $f(n) = g(n)$ (tanpa heuristik)
- GBFS:
 - $g(n)$ diabaikan
 - $h(n)$ = heuristik menuju goal
 - $f(n) = h(n)$
- A*:
 - $g(n)$ = jumlah langkah dari awal
 - $h(n)$ = heuristik menuju goal
 - $f(n) = g(n) + h(n)$

Dalam Heuristic.java, terdapat dua pendekatan heuristik yang digunakan. Pendekatan pertama menghitung jumlah kendaraan yang menghalangi jalan keluar dari kendaraan utama. Pendekatan kedua menghitung jarak grid (dalam unit sel) dari kendaraan utama ke pintu keluar. Kedua pendekatan ini tidak pernah melebihi cost sebenarnya karena tidak memperhitungkan kompleksitas penuh pergerakan kendaraan lain. Oleh karena itu, kedua heuristik ini bersifat admissible, yang berarti tidak akan meng-overestimate jarak sesungguhnya ke goal dan tetap menjamin optimalitas ketika digunakan dalam A*.

BAB 3 SOURCE PROGRAM

Program menggunakan bahasa pemrograman Java. Berikut adalah *source code* dari program.

Struktur Repository

Tucil13_13523014_13523138/

```
├── README.md
├── src/
│   ├── Main.java
│   ├── Solver.java
│   ├── Board.java
│   ├── State.java
│   ├── Util.java
│   └── Style.java
├── test/
│   ├── input/
│   └── output/
├── bin/
└── doc/
```

Main.java

```
import java.io.File;
import java.util.Scanner;
public class Main implements Style{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // Header
        System.out.print(CLEAR_SCREEN);
        System.out.println(RUSHHOUR);
        System.out.println();
        System.out.println("Selamat datang di Rush Hour Solver!");
        System.out.println(ITALIC + YELLOW + "Masukkan input menggunakan angka!" + RESET);
        System.out.println();

        // List all .txt files in ../test/
        File testFolder = new File("../test/input");
        File[] txtFiles = testFolder.listFiles((_, name) -> name.endsWith(".txt"));
        if (txtFiles == null || txtFiles.length == 0) {
            System.out.println(YELLOW + "Tidak ada file .txt di folder ../test/. Program berhenti." +
RESET);
            sc.close();
            throw new RuntimeException("Tidak ada file .txt di folder ../test/. Program berhenti.");
        }
        System.out.println(BRIGHT_GREEN + "Daftar file input berhasil dimuat dari folder test!" + RESET);

        // Pilih file
        int fileChoice = 0;
        String filePath = "";
        String statusMessage = ""; // Variabel untuk menyimpan pesan status
        String statusColor = ""; // Variabel untuk warna pesan
        boolean firstLoad = true; // Flag untuk pertama kali menampilkan pilihan
```

```

while (true) {
    if (firstLoad) {
        firstLoad = false; // Tidak perlu clear screen pada tampilan pertama
    } else {
        System.out.print(CLEAR_SCREEN);
        // Tampilkan pesan status jika ada
        if (!statusMessage.isEmpty()) {
            System.out.println(statusColor + statusMessage + RESET + "\n");
        }
    }

    for (int i = 0; i < txtFiles.length; i++) {
        System.out.println(GRAY + "  " + (i + 1) + ". " + RESET + txtFiles[i].getName());
    }
    System.out.println();
    System.out.print("Pilih file input dengan angka (1-" + txtFiles.length + ") " + RESET + ":\n" +
GREEN + ">>> " + RESET);
    String input = sc.nextLine().trim();
    try {
        fileChoice = Integer.parseInt(input);
        if (fileChoice < 1 || fileChoice > txtFiles.length) throw new NumberFormatException();
        filePath = txtFiles[fileChoice - 1].getPath();
        break;
    } catch (NumberFormatException e) {
        statusMessage = "Pilihan salah. Mohon masukkan angka antara 1 dan " + txtFiles.length + ".";
        statusColor = YELLOW;
    }
}

// Pilih algoritma
int algoChoice = 0;
String algorithm = "";
statusMessage = "File konfigurasi berhasil dimuat!";
statusColor = BRIGHT_GREEN;

while (true) {
    System.out.print(CLEAR_SCREEN);
    // Tampilkan pesan status di awal
    System.out.println(statusColor + statusMessage + RESET + "\n");

    System.out.println(GRAY + "  1. " + RESET + "Uniform Cost Search (UCS)");
    System.out.println(GRAY + "  2. " + RESET + "Greedy Best First Search");
    System.out.println(GRAY + "  3. " + RESET + "A* Search");
    System.out.println();
    System.out.print("Pilih algoritma pencarian:\n" + GREEN + ">>> " + RESET);
    String input = sc.nextLine().trim();
    try {
        algoChoice = Integer.parseInt(input);
        if (algoChoice == 1) algorithm = "UCS";
        else if (algoChoice == 2) algorithm = "Greedy";
        else if (algoChoice == 3) algorithm = "A*";
        else throw new NumberFormatException();
        break;
    } catch (NumberFormatException e) {
        statusMessage = "Pilihan salah. Mohon masukkan angka 1, 2, atau 3.";
        statusColor = YELLOW;
    }
}

// Skip memilih heuristic jika algoritma adalah UCS
int heuristicId = 0;
if (!algorithm.equals("UCS")) {
    // Pilih heuristic
    statusMessage = "Algoritma berhasil dipilih: " + algorithm;
    statusColor = BRIGHT_GREEN;

    while (true) {

```

```

        System.out.print(CLEAR_SCREEN);
        // Tampilkan pesan status di awal
        System.out.println(statusColor + statusMessage + RESET + "\n");

        System.out.println(GRAY + " 1. " + RESET + "Jumlah kendaraan yang menghalangi ke pintu
keluar");

        System.out.println(GRAY + " 2. " + RESET + "Jarak ke pintu keluar");
        System.out.println();
        System.out.print("Pilih heuristic:\n" + GREEN + ">> " + RESET);
        String input = sc.nextLine().trim();
        try {
            heuristicId = Integer.parseInt(input);
            if (heuristicId == 1 || heuristicId == 2) {
                break;
            } else throw new NumberFormatException();
        } catch (NumberFormatException e) {
            statusMessage = "Pilihan salah. Mohon masukkan angka 1 atau 2.";
            statusColor = YELLOW;
        }
    }

    // Mulai proses
    System.out.print(CLEAR_SCREEN);
    System.out.println(BRIGHT_GREEN + "Membaca file dan memulai pencarian solusi..." + RESET);
    System.out.println();

    try {
        Board board = Util.loadBoardFromFile(filePath);
        if (board == null) {
            System.out.println(YELLOW + "Board gagal dimuat. Program berhenti." + RESET);
            return;
        }

        Solver solver = new Solver(board, algorithm, heuristicId);
        State solution = solver.solve();
        System.out.println();

        // Penyimpanan solusi jika ada solusi
        if (solution != null) {
            boolean valid = false;
            while (!valid){
                try {
                    System.out.print("Apakah Anda ingin menyimpan solusi? (y/n):\n" + GREEN + ">> " +
RESET);

                    String response = sc.nextLine().trim().toLowerCase();
                    System.out.println();

                    if (response.equals("y")) {
                        System.out.print("Masukkan nama file solusi " + ITALIC + YELLOW + "(tanpa
ekstensi .txt):\n" + RESET + GREEN + ">> " + RESET);
                        String fileName = sc.nextLine().trim();
                        System.out.println();

                        // Jika nama file kosong, gunakan default
                        if (fileName.isEmpty()) {
                            fileName = "solution";
                        }

                        // Simpan solusi ke file
                        Util.writeSolutionToFile(solution, algorithm, heuristicId, fileName);
                        valid = true;
                    } else if (response.equals("n")){
                        valid = true;
                    } else {
                        throw new IllegalArgumentException("Mohon masukkan pilihan 'y' atau 'n'.\n");
                    }
                } catch (IllegalArgumentException e) {

```

```

        System.out.println(YELLOW + "Pilihan salah. " + e.getMessage() + RESET);
    }
}
} catch (Exception e) {
    System.out.println(YELLOW + "Terjadi kesalahan: " + e.getMessage() + RESET);
} finally {
    sc.close();
    System.out.println(BOLD_GREEN + "\nSampai jumpa di compile selanjutnya!" + RESET);
}
}
}

```

Solver.java

```

import java.util.*;

public class Solver implements Style{
    Board initial;
    String algorithm;
    int heuristicId;

    public Solver(Board board, String algorithm, int heuristicId) {
        this.initial = board;
        this.algorithm = algorithm;
        this.heuristicId = heuristicId;
    }

    public State solve() {
        long startTime = System.nanoTime();

        PriorityQueue<State> queue = new PriorityQueue<>(Comparator.comparingInt(s ->
s.getTotalCost(algorithm)));
        Set<String> visited = new HashSet<>();

        State start = new State(initial, 0, null, null, Heuristic.evaluate(initial, heuristicId));
        queue.add(start);

        int visitedCount = 0;

        while (!queue.isEmpty()) {
            State current = queue.poll();
            visitedCount++;

            if (Util.isSolved(current.board)) {
                long endTime = System.nanoTime();
                Util.printSolution(current);
                System.out.println(ITALIC + YELLOW + "Jumlah node yang dikunjungi: " + visitedCount + RESET);
                System.out.printf(ITALIC + YELLOW + "Waktu eksekusi: %.4f ms%n" + RESET, (endTime - startTime) /
1e6);
                return current;
            }

            String hash = Util.hash(current.board);
            if (visited.contains(hash)) continue;
            visited.add(hash);

            List<State> successors = Util.getSuccessors(current, algorithm, heuristicId);
            queue.addAll(successors);
        }
        Util.printSolution(start);
        System.out.println(RED + "Tidak ditemukan solusi" + RESET);
        return null;
    }
}

```

```
}
```

Board.java

```
import java.util.Arrays;

public class Board implements Style{
    public char[][] grid;
    public int rows, cols;
    public int exitRow, exitCol;

    public Board(int r, int c) {
        this.rows = r;
        this.cols = c;
        this.grid = new char[r][c];
    }

    public Board clone() {
        Board copy = new Board(rows, cols);
        for (int i = 0; i < rows; i++)
            copy.grid[i] = Arrays.copyOf(grid[i], cols);
        copy.exitRow = this.exitRow;
        copy.exitCol = this.exitCol;
        return copy;
    }

    public void printBoard(String highlightLabel) {
        // Border atas (dengan kemungkinan exit di atas)
        System.out.print("  +");
        for (int j = 0; j < cols; j++) {
            if (exitRow == -1 && exitCol == j) {
                System.out.print(GREEN + "K" + RESET + "-");
            } else {
                System.out.print("--");
            }
        }
        System.out.println("+");

        for (int i = 0; i < rows; i++) {
            // Border kiri (dengan kemungkinan exit di kiri)
            if (exitCol == -1 && exitRow == i) {
                System.out.print(GREEN + "K" + RESET);
            } else {
                System.out.print(" |");
            }

            // Isi grid
            for (int j = 0; j < cols; j++) {
                char ch = grid[i][j];
                if (ch == 'P') {
                    System.out.print(RED + ch + RESET + " ");
                } else if (("" + ch).equals(highlightLabel)) {
                    System.out.print(BLUE + ch + RESET + " ");
                } else {
                    System.out.print(ch + " ");
                }
            }

            // Border kanan (dengan kemungkinan exit di kanan)
            if (exitCol == cols && exitRow == i) {
                System.out.print(GREEN + "K" + RESET);
            } else {
                System.out.print("|");
            }
            System.out.println();
        }

        // Border bawah (dengan kemungkinan exit di bawah)
        System.out.print("  +");
    }
}
```

```

        for (int j = 0; j < cols; j++) {
            if (exitRow == rows && exitCol == j) {
                System.out.print(GREEN + "K" + RESET + "--");
            } else {
                System.out.print("--");
            }
        }
        System.out.println("+");
        System.out.println();
    }
}

```

State.java

```

public class State implements Comparable<State> {
    public Board board;
    public int cost;
    public String move;
    public State parent;
    public int heuristic;

    public State(Board b, int cost, String move, State parent, int h) {
        this.board = b;
        this.cost = cost;
        this.move = move;
        this.parent = parent;
        this.heuristic = h;
    }

    public int getTotalCost(String algo) {
        if (algo.equals("UCS")) return cost;
        if (algo.equals("Greedy")) return heuristic;
        return cost + heuristic; // A*
    }

    @Override
    public int compareTo(State other) {
        return Integer.compare(this.getTotalCost("A*"), other.getTotalCost("A*"));
    }
}

```

Util.java

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;

public class Util implements Style{

    public static boolean isSolved(Board board) {
        int ex = board.exitRow;
        int ey = board.exitCol;
        char[][] grid = board.grid;

        int rows = board.rows;
        int cols = board.cols;
    }
}

```



```

// CASE: Horizontal goal (K sejajar baris dengan P)
if (ex >= 0 && ex < rows) {
    for (int j = 0; j < cols; j++) {
        if (grid[ex][j] == 'P') {
            // Cari ujung kanan P
            int end = j;
            while (end + 1 < cols && grid[ex][end + 1] == 'P') end++;

            // Cek arah ke exit (kiri atau kanan)
            // Exit di kiri
            if (ey == -1){
                // Pastikan semua cell antara K dan P kosong
                for (int k = 0; k < j; k++) {
                    if (grid[ex][k] != '.') return false;
                }
                return true;
            }

            // Exit di kanan (ey >= cols)
            // Pastikan semua cell antara P dan K kosong
            for (int k = end + 1; k < ey; k++) {
                if (k < cols && grid[ex][k] != '.') return false;
            }
            // Akhirnya, cek apakah pintu keluar ada tepat di ey
            if (ey == end + 1 || ey > end) {
                return true;
            }
        }
    }
}

// CASE: Vertical goal (K sejajar kolom dengan P)
if (ey >= 0 && ey < cols) {
    for (int i = 0; i < rows; i++) {
        if (grid[i][ey] == 'P') {
            // Cari ujung bawah P
            int end = i;
            while (end + 1 < rows && grid[end + 1][ey] == 'P') end++;

            // Cek arah ke exit (atas atau bawah)
            // Exit di atas
            if (ex < 0) {
                // Pastikan semua cell di atas P kosong
                for (int k = i - 1; k >= 0; k--) {
                    if (grid[k][ey] != '.') return false;
                }
                // P harus menempel di baris paling atas
                if (i == 0) return true;
            }

            // Exit di bawah (// ex >= rows )
            // Pastikan semua cell di bawah P kosong
            for (int k = end + 1; k < rows; k++) {
                if (grid[k][ey] != '.') return false;
            }
            // P harus menempel di baris paling bawah
            if (end == rows - 1) return true;
        }
    }
}

return false;
}

public static String hash(Board board) {

```

```

        StringBuilder sb = new StringBuilder();
        for (char[] row : board.grid)
            sb.append(row);
        return sb.toString();
    }

    public static void printSolution(State state) {
        LinkedList<State> path = new LinkedList<>();
        while (state != null) {
            path.addFirst(state);
            state = state.parent;
        }

        int moveNum = 0;
        // Print langkah-langkah
        for (State s : path) {
            if (moveNum == 0) {
                System.out.println("Papan Awal:");
            } else {
                System.out.printf("Gerakan %d: %s\n", moveNum, s.move);
            }

            String highlight = (s.move != null) ? s.move.split("-")[0] : "";
            s.board.printBoard(highlight);
            moveNum++;
        }
        // Print langkah terakhir dengan P keluar dari board
        if (!path.isEmpty() && isSolved(path.getLast().board)) {
            System.out.println("Papan Akhir:");
            printExitPath(path.getLast().board);
        }
    }

    private static void printExitPath(Board board) {
        // Border atas
        System.out.print("  +");
        for (int j = 0; j < board.cols; j++) {
            if (board.exitRow == -1 && board.exitCol == j) {
                System.out.print(GREEN + "K" + RESET + "-");
            } else {
                System.out.print("--");
            }
        }
        System.out.println("+");

        // Cari posisi P
        int pRow = -1;
        int pCol = -1;
        int pSize = 0;
        boolean isHorizontal = false;

        for (int i = 0; i < board.rows; i++) {
            for (int j = 0; j < board.cols; j++) {
                if (board.grid[i][j] == 'P') {
                    if (pRow == -1) {
                        pRow = i;
                        pCol = j;
                    }
                    pSize++;
                }
            }
        }

        // Tentukan orientasi dan ukuran P
        if (pRow >= 0 && pCol >= 0) {
            // Cek orientasi horizontal
            if (pCol + 1 < board.cols && board.grid[pRow][pCol + 1] == 'P') {
                isHorizontal = true;
            }
        }
    }

```

```

        pSize = 0;
        for (int j = pCol; j < board.cols && board.grid[pRow][j] == 'P'; j++) {
            pSize++;
        }
    }
    // Cek orientasi vertikal
    else if (pRow + 1 < board.rows && board.grid[pRow + 1][pCol] == 'P') {
        isHorizontal = false;
        pSize = 0;
        for (int i = pRow; i < board.rows && board.grid[i][pCol] == 'P'; i++) {
            pSize++;
        }
    }
}

// Menentukan arah exit dan rentang jalur
boolean isHorizontalExit = (board.exitRow >= 0 && board.exitRow < board.rows);
int endPRow = isHorizontal ? pRow : pRow + pSize - 1;
int endPCol = isHorizontal ? pCol + pSize - 1 : pCol;

// Print isi
for (int i = 0; i < board.rows; i++) {
    // Border kiri
    if (board.exitCol == -1 && board.exitRow == i) {
        System.out.print(GREEN + "K" + RESET + " |");
    } else {
        System.out.print(" |");
    }

    // Isi grid
    for (int j = 0; j < board.cols; j++) {
        char cell = board.grid[i][j];

        // Piece P
        if (cell == 'P') {
            System.out.print(BLUE + ". " + RESET);
        }
        // Jalur ke exit
        else if (pRow != -1 && ((isHorizontalExit && i == pRow && j > endPCol && j <= board.exitCol) ||
            (!isHorizontalExit && j == board.exitCol &&
            (board.exitRow < 0 && i < pRow) || (board.exitRow >= board.rows && i > endPRow)))) {
            System.out.print(BLUE + ". " + RESET);
        }
        else {
            System.out.print(cell + " ");
        }
    }

    // Border kanan
    if (board.exitCol == board.cols && board.exitRow == i) {
        System.out.print(GREEN + "K" + RESET);
    } else {
        System.out.print("|");
    }
    System.out.println();
}

// Border bawah
System.out.print(" +");
for (int j = 0; j < board.cols; j++) {
    if (board.exitRow == board.rows && board.exitCol == j) {
        System.out.print(GREEN + "K" + RESET + "-");
    } else {
        System.out.print("--");
    }
}

```

```

    }
    System.out.println("+");
    System.out.println();
}

public static int countBlockingCars(Board board) {
    int rows = board.rows;
    int cols = board.cols;
    char[][] grid = board.grid;

    for (int i = 0; i < rows; i++) {
        int pStart = -1, pEnd = -1;
        for (int j = 0; j < cols; j++) {
            if (grid[i][j] == 'P') {
                if (pStart == -1) pStart = j;
                pEnd = j;
            }
        }

        if (pStart != -1 && pEnd != -1) {
            int count = 0;
            for (int j = pEnd + 1; j < cols; j++) {
                if (grid[i][j] != '.' && grid[i][j] != 'K') {
                    count++;
                }
            }
            return count;
        }
    }

    return Integer.MAX_VALUE; // fallback jika tidak ditemukan P
}

public static int distanceToExit(Board board) {
    int rows = board.rows;
    int cols = board.cols;
    char[][] grid = board.grid;

    for (int i = 0; i < rows; i++) {
        int pEnd = -1;
        int kIndex = -1;

        for (int j = 0; j < cols; j++) {
            if (grid[i][j] == 'P') {
                pEnd = j;
            }
            if (grid[i][j] == 'K') {
                kIndex = j;
            }
        }

        if (pEnd != -1 && kIndex != -1 && kIndex > pEnd) {
            return kIndex - pEnd - 1;
        }
    }

    return Integer.MAX_VALUE; // fallback
}

public static Board loadBoardFromFile(String path) {
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {

        /* Edge case handling */
        // Baca baris pertama: ukuran board
        String checkSizeLine = br.readLine();
        if (checkSizeLine == null)
            throw new IllegalArgumentException("Format file tidak valid! File kosong.");
    }
}

```

```

        String[] size = checkSizeLine.trim().split("\\s+");
        if (size.length != 2)
            throw new IllegalArgumentException("Format file tidak valid! Baris pertama harus memiliki 2
nilai: rows dan cols. Diberikan: " + Arrays.toString(size));

        int rows, cols;
        try {
            rows = Integer.parseInt(size[0]);
            cols = Integer.parseInt(size[1]);
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("Format file tidak valid! Baris pertama harus berupa
angka, sedangkan diberikan: " + Arrays.toString(size));
        }

        // Baca baris kedua: jumlah kendaraan
        String checkPiecesLine = br.readLine();
        if (checkPiecesLine == null)
            throw new IllegalArgumentException("Format file tidak valid. Baris kedua (jumlah kendaraan)
tidak ditemukan.");

        int pieceCount;
        try {
            pieceCount = Integer.parseInt(checkPiecesLine.trim());
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("Format file tidak valid. Baris kedua (jumlah kendaraan)
harus berupa angka, sedangkan diberikan: " + checkPiecesLine);
        }

        /* Baca Grid */
        Board board = new Board(rows, cols);
        board.exitRow = -1;
        board.exitCol = -1;

        List<String> gridLines = new ArrayList<>();
        String line;
        while ((line = br.readLine()) != null) {
            line = line.stripTrailing();
            if (!line.isEmpty()) gridLines.add(line);
        }

        /* Cek 'K' muncul di atas/bawah grid */
        if (gridLines.size() > rows){
            // K di atas grid
            if (gridLines.get(0).contains("K")){
                board.exitRow = -1;
                board.exitCol = gridLines.get(0).indexOf('K');
                gridLines.remove(0);
            }

            // K di bawah grid
            if (gridLines.get(gridLines.size() - 1).contains("K")){
                // Handle duplicate 'K'
                if (board.exitCol != -1 || board.exitRow != -1)
                    throw new IllegalArgumentException("Format file tidak valid! 'K' duplikat ditemukan.");
                board.exitRow = rows;
                board.exitCol = gridLines.get(gridLines.size() - 1).indexOf('K');
                gridLines.remove(gridLines.size() - 1);
            }
        }
        // Validasi jumlah baris setelah pengecekan K di atas bawah
        if (gridLines.size() != rows)
            throw new IllegalArgumentException("Format file tidak valid! Jumlah baris pada papan (" +
gridLines.size() + ") tidak sesuai dengan ukuran yang dideklarasikan (" + rows + ").");

        for (int i = 0; i < rows; i++){
            String rowline = gridLines.get(i);

```

```

        // K di kiri
        if (rowLine.length() > 0 && rowLine.charAt(0) == 'K'){
            // Handle duplicate 'K'
            if (board.exitCol != -1 || board.exitRow != -1)
                throw new IllegalArgumentException("Format file tidak valid! 'K' duplikat ditemukan.");
            board.exitRow = i;
            board.exitCol = -1;
            rowLine = rowLine.substring(1);
        }

        // K di kanan
        if (rowLine.length() > cols && rowLine.charAt(cols) == 'K'){
            // Handle duplicate 'K'
            if (board.exitCol != -1 || board.exitRow != -1)
                throw new IllegalArgumentException("Format file tidak valid! 'K' duplikat
ditemukan.");
            board.exitRow = i;
            board.exitCol = cols;
        }

        // Validasi panjang baris setelah pengecekan K di kiri kanan
        if (!(rowLine.length() == cols || (rowLine.length() == cols + 1 && rowLine.charAt(cols) ==
'K'))))
            throw new IllegalArgumentException("Format file tidak valid! Panjang baris ke-" + (i+1)
+ " (" + rowLine.length() + ") tidak sesuai dengan jumlah kolom (" + cols + ").");

        // Isi grid
        for (int j = 0; j < cols; j++) {
            if (j < rowLine.length()) {
                char ch = rowLine.charAt(j);

                // jika K di dalam board (sudah dicek di pinggir dan atas bawah)
                if (ch == 'K'){
                    throw new IllegalArgumentException("Format file tidak valid! 'K' ditemukan dalam
papan, sedangkan hanya boleh di dinding papan.");
                } else {
                    board.grid[i][j] = ch;
                }
            } else {
                board.grid[i][j] = '.';
            }
        }

        if (board.exitRow == -1 && board.exitCol == -1)
            throw new IllegalArgumentException("Format file tidak valid! 'K' tidak ditemukan dalam
board.");

        System.out.println("Board berhasil dimuat!");
        // board.printBoard("");

        System.out.println("Posisi K: (" + board.exitRow + ", " + board.exitCol + ")");
        System.out.println("Jumlah kendaraan: " + pieceCount);
        System.out.println("Ukuran papan: " + rows + " x " + cols);
        System.out.println();

        return board;
    } catch (IllegalArgumentException e) {
        System.err.println(e.getMessage());
    } catch (IOException e) {
        System.err.println("Gagal membaca file: " + e.getMessage());
    } catch (Exception e) {
        System.err.println("Format file salah: " + e.getMessage());
    }
    return null;
}

```

```

public static List<State> getSuccessors(State current, String algorithm, int heuristicId) {
    List<State> successors = new ArrayList<>();
    Board board = current.board;

    Map<Character, Piece> pieces = detectPieces(board);

    for (Piece piece : pieces.values()) {
        // Geser maju
        for (int d = 1; canMove(piece, board, d); d++) {
            Board newBoard = movePiece(piece, board, d);
            int cost = current.cost + d;
            int h = Heuristic.evaluate(newBoard, heuristicId);
            String move = piece.label + "-" + (piece.horizontal ? "kanan" : "bawah");
            successors.add(new State(newBoard, cost, move, current, h));
        }

        // Geser mundur
        for (int d = -1; canMove(piece, board, d); d--) {
            Board newBoard = movePiece(piece, board, d);
            int cost = current.cost + Math.abs(d);
            int h = Heuristic.evaluate(newBoard, heuristicId);
            String move = piece.label + "-" + (piece.horizontal ? "kiri" : "atas");
            successors.add(new State(newBoard, cost, move, current, h));
        }
    }

    return successors;
}

private static Map<Character, Piece> detectPieces(Board board) {
    Map<Character, Piece> pieces = new HashMap<>();

    for (int i = 0; i < board.rows; i++) {
        for (int j = 0; j < board.cols; j++) {
            char c = board.grid[i][j];
            if (c != '.' && c != 'K' && !pieces.containsKey(c)) {
                // Cek horizontal
                if (j + 1 < board.cols && board.grid[i][j + 1] == c) {
                    int size = 1;
                    while (j + size < board.cols && board.grid[i][j + size] == c)
                        size++;
                    pieces.put(c, new Piece(c, i, j, size, true));
                }
                // Cek vertikal
                else if (i + 1 < board.rows && board.grid[i + 1][j] == c) {
                    int size = 1;
                    while (i + size < board.rows && board.grid[i + size][j] == c)
                        size++;
                    pieces.put(c, new Piece(c, i, j, size, false));
                }
            }
        }
    }

    return pieces;
}

private static boolean canMove(Piece piece, Board board, int delta) {
    int r = piece.row, c = piece.col;
    if (piece.horizontal) {
        if (delta > 0) {
            int end = c + piece.size - 1;
            if (end + delta >= board.cols) return false;
            for (int i = 1; i <= delta; i++) {
                if (board.grid[r][end + i] != '.') return false;
            }
        } else {
            if (c + delta < 0) return false;
        }
    }
}

```

```

        for (int i = -1; i >= delta; i--) {
            if (board.grid[r][c + i] != '.') return false;
        }
    } else {
        if (delta > 0) {
            int end = r + piece.size - 1;
            if (end + delta >= board.rows) return false;
            for (int i = 1; i <= delta; i++) {
                if (board.grid[end + i][c] != '.') return false;
            }
        } else {
            if (r + delta < 0) return false;
            for (int i = -1; i >= delta; i--) {
                if (board.grid[r + i][c] != '.') return false;
            }
        }
    }
    return true;
}

private static Board movePiece(Piece piece, Board board, int delta) {
    Board newBoard = board.clone();
    int r = piece.row, c = piece.col;
    char ch = piece.label;

    // Kosongkan posisi lama
    if (piece.horizontal) {
        for (int i = 0; i < piece.size; i++)
            newBoard.grid[r][c + i] = '.';
        int newCol = c + delta;
        for (int i = 0; i < piece.size; i++)
            newBoard.grid[r][newCol + i] = ch;
    } else {
        for (int i = 0; i < piece.size; i++)
            newBoard.grid[r + i][c] = '.';
        int newRow = r + delta;
        for (int i = 0; i < piece.size; i++)
            newBoard.grid[newRow + i][c] = ch;
    }

    return newBoard;
}

public static void writeSolutionToFile(State state, String algorithm, int heuristicId, String fileName)
{
    try {
        // Buat folder output jika belum ada
        File outputDir = new File("../test/output");
        if (!outputDir.exists()) {
            outputDir.mkdir();
        }

        // Hilangkan ekstensi .txt jika ada pada nama file dari input pengguna
        if (fileName.toLowerCase().endsWith(".txt")) {
            fileName = fileName.substring(0, fileName.length() - 4);
        }
        String filename = "../test/output/" + fileName + ".txt";

        // Buat file writer
        PrintWriter writer = new PrintWriter(new FileWriter(filename));

        writer.println("Rush Hour Solution");
        writer.println("Algorithm: " + algorithm);
        if (!algorithm.equals("UCS")) {
            writer.println("Heuristic: " + (heuristicId == 1 ? "Jumlah kendaraan yang menghalangi" :
"Jarak ke pintu keluar"));
        }
    }
}

```



```

        writer.println();

        // Rekonstruksi path solusi
        LinkedList<State> path = new LinkedList<>();
        while (state != null) {
            path.addFirst(state);
            state = state.parent;
        }

        // Write langkah-langkah solusi
        int moveNum = 0;
        for (State s : path) {
            if (moveNum == 0) {
                writer.println("Papan Awal:");
            } else {
                writer.printf("Gerakan %d: %s\n", moveNum, s.move);
            }

            // Representasi papan sebagai string
            writer.println(boardToString(s.board, (s.move != null) ? s.move.split("-")[0] : ""));
            moveNum++;
        }

        // Tambahkan papan akhir
        if (!path.isEmpty() && isSolved(path.getLast().board)) {
            writer.println("Papan Akhir:");
            writer.println(boardToStringWithExitPath(path.getLast().board));
        }

        writer.close();

        System.out.println(Style.BRIGHT_GREEN + "Solusi berhasil disimpan ke " + filename +
            Style.RESET);
        return;
    } catch (IOException e) {
        System.out.println(Style.YELLOW + "Gagal menyimpan solusi: " + e.getMessage() + Style.RESET);
    }
}

// Tambahkan method untuk mengkonversi board ke string tanpa ANSI color
private static String boardToString(Board board, String highlightLabel) {
    StringBuilder sb = new StringBuilder();

    // Border atas (dengan kemungkinan exit di atas)
    sb.append("  +");
    for (int j = 0; j < board.cols; j++) {
        if (board.exitRow == -1 && board.exitCol == j) {
            sb.append("K-");
        } else {
            sb.append("--");
        }
    }
    sb.append("+\n");

    for (int i = 0; i < board.rows; i++) {
        // Border kiri (dengan kemungkinan exit di kiri)
        if (board.exitCol == -1 && board.exitRow == i) {
            sb.append("K |");
        } else {
            sb.append("  |");
        }

        // Isi grid
        for (int j = 0; j < board.cols; j++) {
            sb.append(board.grid[i][j] + " ");
        }

        // Border kanan (dengan kemungkinan exit di kanan)
    }
}

```

```

        if (board.exitCol == board.cols && board.exitRow == i) {
            sb.append("K");
        } else {
            sb.append("|");
        }
        sb.append("\n");
    }

    // Border bawah (dengan kemungkinan exit di bawah)
    sb.append(" +");
    for (int j = 0; j < board.cols; j++) {
        if (board.exitRow == board.rows && board.exitCol == j) {
            sb.append("K-");
        } else {
            sb.append("--");
        }
    }
    sb.append("+\n\n");

    return sb.toString();
}

// Method untuk membuat string representasi board dengan P diganti titik
private static String boardToStringWithExitPath(Board board) {
    StringBuilder sb = new StringBuilder();

    // Border atas (dengan kemungkinan exit di atas)
    sb.append(" +");
    for (int j = 0; j < board.cols; j++) {
        if (board.exitRow == -1 && board.exitCol == j) {
            sb.append("K-");
        } else {
            sb.append("--");
        }
    }
    sb.append("+\n");

    for (int i = 0; i < board.rows; i++) {
        // Border kiri (dengan kemungkinan exit di kiri)
        if (board.exitCol == -1 && board.exitRow == i) {
            sb.append("K |");
        } else {
            sb.append(" |");
        }

        // Isi grid
        for (int j = 0; j < board.cols; j++) {
            // Ganti 'P' dengan '.'
            if (board.grid[i][j] == 'P') {
                sb.append(". ");
            } else {
                sb.append(board.grid[i][j] + " ");
            }
        }

        // Border kanan (dengan kemungkinan exit di kanan)
        if (board.exitCol == board.cols && board.exitRow == i) {
            sb.append("K");
        } else {
            sb.append("|");
        }
        sb.append("\n");
    }

    // Border bawah (dengan kemungkinan exit di bawah)
    sb.append(" +");
    for (int j = 0; j < board.cols; j++) {
        if (board.exitRow == board.rows && board.exitCol == j) {

```

```

        sb.append("K-");
    } else {
        sb.append("--");
    }
}
sb.append("+\n\n");

return sb.toString();
}
}

```

Style.java

```

public interface Style {
    public static final String RESET = "\033[0m";
    public static final String ITALIC = "\033[3m";
    public static final String GREEN = "\033[32m";
    public static final String BRIGHT_GREEN = "\033[92m";
    public static final String BOLD_GREEN = "\033[1;32m";
    public static final String YELLOW = "\033[33m";
    public static final String RED = "\033[1;31m";
    public static final String BLUE = "\u001B[34m";
    public static final String GRAY = "\033[90m";
    public static final String CLEAR_SCREEN = "\033c";
    public static final String WHITE = "\033[97m";
}

```

BAB 4 PENGUJIAN

Algoritma Uniform Cost Search (UCS)

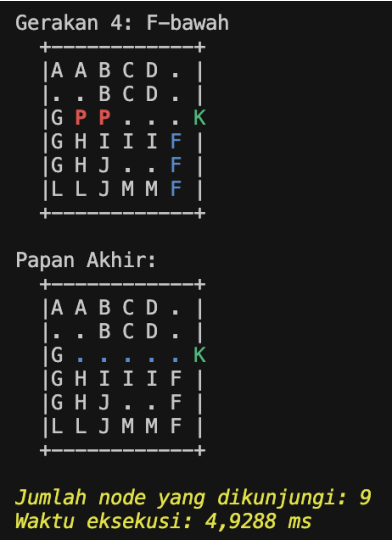
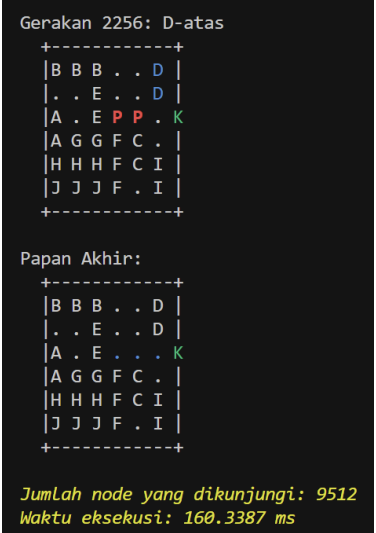
Input	Output
<p>K terletak di atas</p> <pre>test > input > ≡ atas.txt 1 6 6 2 √ 11 3 K 4 AAB..F 5 .PBCDF 6 GP.CDF 7 GH.III 8 GHJ... 9 LLJMM.</pre>	<pre>Gerakan 4: P-atas +--K-----+ . P A A . F . P B C D F . B C D F G H . I I I G H J . . . L L J M M . +-----+ Papan Akhir: +--K-----+ . . A A . F . . B C D F . B C D F G H . I I I G H J . . . L L J M M . +-----+ Jumlah node yang dikunjungi: 602 Waktu eksekusi: 14.4676 ms</pre>
<p>K terletak di bawah</p> <pre>test > input > ≡ bawah1.txt 1 6 6 2 10 3 CUUP.. 4 CABP.. 5 .ABZZZ 6 DDBLLQ 7 ..MM.Q 8 ...HHH 9 K</pre>	<pre>Gerakan 55: P-bawah +-----+ C A . U U Q C A . . . Q . . B Z Z Z D D B . L L . . B P M M H H H P . . +-----K-----+ Papan Akhir: +-----+ C A . U U Q C A . . . Q . . B Z Z Z D D B . L L . . B . M M H H H . . . +-----K-----+ Jumlah node yang dikunjungi: 33240 Waktu eksekusi: 69.6659 ms</pre>
<p>K terletak di bawah (tidak ada solusi)</p> <pre>test > input > ≡ bawah2.txt 1 6 6 2 11 3 AAB..F 4 .PBCDF 5 GP.CDF 6 GH.III 7 GHJ... 8 √ LLJMM. 9 K</pre>	<pre>Papan Awal: +-----+ A A B . . F . P B C D F G P . C D F G H . I I I G H J . . . L L J M M . +--K-----+ Tidak ditemukan solusi</pre>

<p>K terletak di kanan 1</p> <pre>test > input > ≡ kanan1.txt 1 6 6 2 11 3 AAB..F 4 ..BCDF 5 GPPCDFK 6 GH.III 7 GHJ... 8 LLJMM.</pre>	<pre>Gerakan 5: F-bawah +-----+ A A B C D . . . B C D . G P P . . . K G H I I I F G H J . . F L L J M M F +-----+ Papan Akhir: +-----+ A A B C D . . . B C D . G K G H I I I F G H J . . F L L J M M F +-----+ Jumlah node yang dikunjungi: 1041 Waktu eksekusi: 15.8351 ms</pre>
<p>K terletak di kanan 2</p> <pre>test > input > ≡ kanan2.txt 1 6 6 2 9 3 ABBBCD 4 A.E.CD 5 PPEF..K 6 ...FGG 7 HHHF.I 8 .JJJ.I</pre>	<pre>Gerakan 31: D-atas +-----+ . . B B B D . . E . . D A . E P P . K A G G F C . H H H F C I J J J F . I +-----+ Papan Akhir: +-----+ . . B B B D . . E . . D A . E . . K A G G F C . H H H F C I J J J F . I +-----+ Jumlah node yang dikunjungi: 63184 Waktu eksekusi: 123.5667 ms</pre>
<p>K terletak di kiri</p> <pre>test > input > ≡ kiri1.txt 1 6 6 2 8 3 BA 4 .CC.BA 5 K.EDPPA 6 .EDGFF 7 .HHG.. 8 9</pre>	<pre>Gerakan 5: D-bawah +-----+ . E . . B A . E C C B A K. . . P A . . D G F F H H D G +-----+ Papan Akhir: +-----+ . E . . B A . E C C B A K. . . . A . . D G F F H H D G +-----+ Jumlah node yang dikunjungi: 189 Waktu eksekusi: 284.2286 ms</pre>
<p>K duplikat</p>	<pre>Format file tidak valid! 'K' duplikat ditemukan. Board gagal dimuat. Program berhenti.</pre>

<pre> test > input > ≡ duplikat.txt 1 6 6 2 11 3 K 4 AAB..F 5 .PBCDF 6 GP.CDF 7 GH.III 8 GHJ... 9 LLJMM. 10 K </pre>	
<p>K tidak ada</p> <pre> 1 3 3 2 1 3 A.. 4 AP. 5 AP. </pre>	<pre> Format file tidak valid! 'K' tidak ditemukan dalam board. Board gagal dimuat. Program berhenti. </pre>

Algoritma Greedy Best First Search (GBFS)

Input	Output
<p>K terletak di atas</p> <pre>test > input > ≡ atas.txt 1 6 6 2 √ 11 3 K 4 AAB..F 5 .PBCDF 6 GP.CDF 7 GH.III 8 GHJ... 9 LLJMM.</pre>	<p>Heuristik: <i>Jumlah kendaraan yang menghalangi ke pintu keluar</i></p> <pre>Gerakan 10: P-atas +--K-----+ . P . A A F G P . C D F G . B C D F G H B I I I . H J . . . L L J M M . +-----+ Papan Akhir: +--K-----+ . . . A A F G . . C D F G . B C D F G H B I I I . H J . . . L L J M M . +-----+ Jumlah node yang dikunjungi: 90 Waktu eksekusi: 19.0873 ms</pre>
<p>K terletak di bawah</p> <pre>test > input > ≡ bawah1.txt 1 6 6 2 10 3 CUUP.. 4 CABP.. 5 .ABZZZ 6 DDBLLQ 7 ..MM.Q 8 ...HHH 9 K</pre>	<p>Heuristik: <i>Jarak ke pintu keluar</i></p> <pre>Gerakan 1114: P-bawah +-----+ . . B U U Q . A B . . Q . A B Z Z Z C D D . L L C . . P M M H H H P . . +-----K-----+ Papan Akhir: +-----+ . . B U U Q . A B . . Q . A B Z Z Z C D D . L L C . . P M M H H H . . . +-----K-----+ Jumlah node yang dikunjungi: 9786 Waktu eksekusi: 32,8397 ms</pre>
<p>K terletak di bawah (tidak ada solusi)</p> <pre>test > input > ≡ bawah2.txt 1 6 6 2 11 3 AAB..F 4 .PBCDF 5 GP.CDF 6 GH.III 7 GHJ... 8 √ LLJMM. 9 K</pre>	<p>Papan Awal:</p> <pre>+-----+ A A B . . F . P B C D F G P . C D F G H . I I I G H J . . . L L J M M . +-----K-----+ Tidak ditemukan solusi</pre>

<p>K terletak di kanan 1 (dengan heuristik)</p> <pre> test > input > ≡ kanan1.txt 1 6 6 2 11 3 AAB..F 4 ..BCDF 5 GPPCDFK 6 GH.III 7 GHJ... 8 LLJMM. </pre>	<p>Heuristik: <i>Jumlah kendaraan yang menghalangi ke pintu keluar</i></p> 
<p>K terletak di kanan 2 (dengan heuristik Jarak ke pintu keluar)</p> <pre> test > input > ≡ kanan2.txt 1 6 6 2 9 3 ABBBCD 4 A.E.CD 5 PPEF..K 6 ...FGG 7 HHHF.I 8 .JJJ.I </pre>	<p>Heuristik: <i>Jarak ke pintu keluar</i></p> 
<p>K terletak di kiri</p> <pre> test > input > ≡ kiri1.txt 1 6 6 2 8 3 BA 4 .CC.BA 5 K.EDPPA 6 .EDGFF 7 .HHG.. 8 </pre>	<p>Heuristik: <i>Jumlah kendaraan yang menghalangi ke pintu keluar</i></p>

	<pre> Gerakan 9: D-bawah +-----+ . E . . B . . E C C B . K. . . P P . . . D F F A H H D G . A . . . G . A +-----+ Papan Akhir: +-----+ . E . . B . . E C C B . K. D F F A H H D G . A . . . G . A +-----+ Jumlah node yang dikunjungi: 22 Waktu eksekusi: 295.1771 ms </pre>
<p>K duplikat</p> <pre> test > input > ≡ duplikat.txt 1 6 6 2 11 3 K 4 AAB..F 5 .PBCDF 6 GP.CDF 7 GH.III 8 GHJ... 9 LLJMM. 10 K </pre>	<pre> Format file tidak valid! 'K' duplikat ditemukan. Board gagal dimuat. Program berhenti. </pre>
<p>K tidak ada</p> <pre> 1 3 3 2 1 3 A.. 4 AP. 5 AP. </pre>	<pre> Format file tidak valid! 'K' tidak ditemukan dalam board. Board gagal dimuat. Program berhenti. </pre>

Algoritma A* Search

Input	Output
K terletak di atas	Heuristik: <i>Jumlah kendaraan yang menghalangi ke pintu keluar</i>

<pre> test > input > ≡ atas.txt 1 6 6 2 √ 11 3 K 4 AAB..F 5 .PBCDF 6 GP.CDF 7 GH.III 8 GHJ... 9 LLJMM. </pre>	<pre> Gerakan 3: P-atas +-----+ . P A A . F . P B C D F G . B C D F G H . I I I G H J . . . L L J M M . +-----+ Papan Akhir: +-----+ . . A A . F . . B C D F G . B C D F G H . I I I G H J . . . L L J M M . +-----+ Jumlah node yang dikunjungi: 191 Waktu eksekusi: 11,9281 ms </pre>
<p>K terletak di bawah</p> <pre> test > input > ≡ bawah1.txt 1 6 6 2 10 3 CUUP.. 4 CABP.. 5 .ABZZZ 6 DDBLLQ 7 ..MM.Q 8 ...HHH 9 K </pre>	<p>Heuristik: <i>Jarak ke pintu keluar</i></p> <pre> Gerakan 55: P-bawah +-----+ C A . U U Q C A . . . Q . . B Z Z Z D D B . L L . . B P M M H H H P . . +-----+ +-----K-----+ Papan Akhir: +-----+ C A . U U Q C A . . . Q . . B Z Z Z D D B . L L . . B . M M H H H . . . +-----+ +-----K-----+ Jumlah node yang dikunjungi: 33240 Waktu eksekusi: 173.4486 ms </pre>
<p>K terletak di bawah (tidak ada solusi)</p> <pre> test > input > ≡ bawah2.txt 1 6 6 2 11 3 AAB..F 4 .PBCDF 5 GP.CDF 6 GH.III 7 GHJ... 8 √ LLJMM. 9 K </pre>	<pre> Papan Awal: +-----+ A A B . . F . P B C D F G P . C D F G H . I I I G H J . . . L L J M M . +-----+ +-----K-----+ Tidak ditemukan solusi </pre>
<p>K terletak di kanan 1</p>	<p>Heuristik: <i>Jumlah kendaraan yang menghalangi ke pintu keluar</i></p>

```
test > input > ≡ kanan1.txt
1   6 6
2   11
3   AAB..F
4   ..BCDF
5   GPPCDFK
6   GH.III
7   GHJ...
8   LLJMM.
```

Gerakan 4: C-atas

```
+-----+
| A A B C D . |
| . . B C D . |
| G P P . . K |
| G H I I I F |
| G H J . . F |
| L L J M M F |
+-----+
```

Papan Akhir:

```
+-----+
| A A B C D . |
| . . B C D . |
| G . . . . K |
| G H I I I F |
| G H J . . F |
| L L J M M F |
+-----+
```

Jumlah node yang dikunjungi: 283
Waktu eksekusi: 11,0602 ms

K terletak di kanan 2

```
test > input > ≡ kanan2.txt
1   6 6
2   9
3   ABBBCD
4   A.E.CD
5   PPEF..K
6   ...FGG
7   HHHF.I
8   .JJJ.I
```

Heuristik: Jarak ke pintu keluar

Gerakan 31: D-atas

```
+-----+
| . . B B B D |
| . . E . . D |
| A . E P P . K |
| A G G F C . |
| H H H F C I |
| J J J F . I |
+-----+
```

Papan Akhir:

```
+-----+
| . . B B B D |
| . . E . . D |
| A . E . . K |
| A G G F C . |
| H H H F C I |
| J J J F . I |
+-----+
```

Jumlah node yang dikunjungi: 63184
Waktu eksekusi: 91,5871 ms

K terletak di kiri

```
test > input > ≡ kiri1.txt
1   6 6
2   8
3   ....BA
4   .CC.BA
5   K.EDPPA
6   .EDGFF
7   .HHG..
8   .....
```

Heuristik: Jumlah kendaraan yang menghalangi ke
pintu keluar

	<pre>Gerakan 5: D-bawah +-----+ . E . . B A . E C C B A K. . . P P A . . D G F F H H D G +-----+ Papan Akhir: +-----+ . E . . B A . E C C B A K. . . . A . . D G F F H H D G +-----+ Jumlah node yang dikunjungi: 190 Waktu eksekusi: 339.1612 ms</pre>
<p>K duplikat</p> <pre>test > input > ≡ duplikat.txt 1 6 6 2 11 3 K 4 AAB..F 5 .PBCDF 6 GP.CDF 7 GH.III 8 GHJ... 9 LLJMM. 10 K</pre>	<pre>Format file tidak valid! 'K' duplikat ditemukan. Board gagal dimuat. Program berhenti.</pre>
<p>K tidak ada</p> <pre>1 3 3 2 1 3 A.. 4 AP. 5 AP. </pre>	<pre>Format file tidak valid! 'K' tidak ditemukan dalam board. Board gagal dimuat. Program berhenti.</pre>

BAB 5 HASIL ANALISIS & KESIMPULAN

Uniform Cost Search (UCS)

- Kompleksitas waktu dari UCS adalah $O(b^d)$, di mana b adalah jumlah kemungkinan pergerakan per state, dan d adalah kedalaman solusi optimal.
- UCS mengeksplorasi seluruh simpul berdasarkan urutan cost terkecil tanpa memperhitungkan heuristik, sehingga ia dapat mengeksplorasi banyak state yang tidak mendekati solusi.
- Seperti halnya BFS, UCS akan melakukan ekspansi melebar dan menyimpan semua state yang pernah dikunjungi dalam `visited`, sehingga kompleksitas ruang menjadi $O(b^d)$.
- Pada kasus-kasus yang lebih kompleks, UCS dapat mencapai waktu eksekusi yang lebih lama dibanding menggunakan A*, terutama karena eksplorasi yang “buta” terhadap banyak simpul tidak relevan.

Greedy Best First Search (GBFS)

- Kompleksitas waktu dari GBFS adalah $O(b^m)$, dengan m adalah panjang path yang ditemukan. GBFS hanya mempertimbangkan $h(n)$ dan langsung mengejar state yang tampak paling dekat ke goal.
- Kompleksitas ruang dapat menjadi besar karena simpul yang diekspansi bisa masuk antrian walaupun tidak optimal. Namun pada beberapa kasus, jumlah node yang dieksplorasi lebih sedikit dari UCS.
- Dalam eksperimen, GBFS terlihat paling cepat dalam menemukan solusi, namun terkadang menghasilkan solusi yang lebih panjang (kurang optimal).

A* Search

- Kompleksitas waktu dari A* adalah $O(b^d)$ sama seperti UCS. Namun, karena menggunakan heuristik untuk memprioritaskan ekspansi ke arah goal, A* menjadi lebih efisien.
- Kompleksitas ruang A* cukup tinggi karena ia menyimpan semua simpul terbuka dalam priority queue dan semua state yang sudah dikunjungi (`visited`) untuk keperluan backtracking dan pencegahan eksplorasi ulang.
- Dalam percobaan, A* memberikan performa yang paling seimbang. Solusi cepat ditemukan dan tetap optimal, khususnya saat heuristik yang digunakan bersifat admissible dan informatif.

BAB 6 IMPLEMENTASI BONUS

Bonus yang diimplementasikan adalah tambahan heuristik alternatif sebanyak 2 jenis. Heuristik hanya dapat diimplementasikan pada pencarian dengan algoritma Greedy Best First Search (GBFS) dan A*.

Tampilan CLI setelah memilih algoritma GBFS	Tampilan CLI setelah memilih algoritma A* Search
<pre>Algoritma berhasil dipilih: Greedy 1. Jumlah kendaraan yang menghalangi ke pintu keluar 2. Jarak ke pintu keluar Pilih heuristic: >> </pre>	<pre>Algoritma berhasil dipilih: A* 1. Jumlah kendaraan yang menghalangi ke pintu keluar 2. Jarak ke pintu keluar Pilih heuristic: >> </pre>

Heuristik 1: Jumlah Kendaraan yang Menghalangi ke Pintu Keluar

Heuristik ini menghitung berapa banyak kendaraan yang secara langsung menghalangi kendaraan utama untuk keluar dari papan. Implementasinya berfokus pada baris kendaraan utama dan memeriksa setiap grid ke arah pintu keluar dan menghitung berapa kendaraan berbeda yang menghalangi jalur lurus tersebut. Heuristik ini cukup informatif karena langsung fokus pada penghalang yang perlu dipindahkan.

Heuristik 2: Jarak ke Pintu Keluar

Heuristik ini menghitung jarak (dalam satuan grid) dari ujung kendaraan utama ke pintu keluar, tanpa mempertimbangkan ada tidaknya penghalang di antara keduanya. Secara perhitungan, heuristik ini lebih sederhana, tetapi tidak memperhitungkan kendaraan lain yang bisa sangat mempengaruhi jumlah langkah keseluruhan. Heuristik ini juga dapat menyesatkan algoritma GBFS, karena kondisi papan tidak memperbolehkan pergerakan langsung walaupun jaraknya pendek.

Analisis Berdasarkan Pengujian

Untuk GBFS, perbedaan hasil antara kedua heuristik sangat signifikan. Dalam beberapa pengujian, penggunaan heuristik kedua menyebabkan algoritma terjebak dalam jalur yang tidak efektif, karena mengejar jarak pendek tetapi mengabaikan banyaknya kendaraan yang menghalangi. Berikut merupakan konfigurasi awal papan dan hasil menggunakan kedua heuristik untuk kasus **Greedy Best First Search** pada test case **kanan1.txt**.

Papan Awal:

```

+-----+
| A A B . . F |
| . . B C D F |
| G P P C D F K |
| G H . I I I |
| G H J . . . |
| L L J M M . |
+-----+

```

Konfigurasi papan awal

Heuristik 1: Jumlah Kendaraan yang Menghalangi ke Pintu Keluar	Heuristik 2: Jarak ke Pintu Keluar
<p>Gerakan 4: F-bawah</p> <pre> +-----+ A A B C D . . . B C D . G P P . . K G H I I I F G H J . . F L L J M M F +-----+ </pre> <p>Papan Akhir:</p> <pre> +-----+ A A B C D . . . B C D . G K G H I I I F G H J . . F L L J M M F +-----+ </pre> <p>Jumlah node yang dikunjungi: 9 Waktu eksekusi: 13.3518 ms</p>	<p>Gerakan 30: C-atas</p> <pre> +-----+ A A B C D . . . B C D . G P P . . K G H I I I F G H J . . F L L J M M F +-----+ </pre> <p>Papan Akhir:</p> <pre> +-----+ A A B C D . . . B C D . G K G H I I I F G H J . . F L L J M M F +-----+ </pre> <p>Jumlah node yang dikunjungi: 100 Waktu eksekusi: 24.5092 ms</p>

Hasil ekstrem menunjukan solusi heuristik kedua bisa mencapai lebih dari 100 langkah. Pada kasus-kasus yang lebih kompleks, solusi dapat mencapai berkali-kali lipat daripada heuristik pertama. Sehingga, penggunaan heuristik pertama memberi jalur yang jauh lebih dekat ke optimal.

Untuk pencarian A*, kedua heuristik menghasilkan solusi yang cukup optimal. Hal ini dikarenakan pencarian A* tetap mempertimbangkan $g(n)$. Heuristik pertama cenderung menghasilkan solusi dengan

jumlah langkah yang lebih sedikit. Dalam pencarian ini, dapat disimpulkan bahwa kedua heuristik sudah admissible.

LAMPIRAN

A. Github

https://github.com/andhikalucas/Tucil3_13523014_13523138

B. Tabel Pemeriksaan

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4	Program dapat membaca masukan berkas .txt dan menyimpan solusi berupa print board tahap per tahap dalam berkas .txt	✓	
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	✓	

REFERENSI

1. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf)
2. <https://socs.binus.ac.id/2017/08/24/searching-uniform-cost-search/>
3. <https://socs.binus.ac.id/2013/04/23/uninformed-search-dan-informed-search/>