IF2211 Strategi Algoritma
Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding
Laporan Tugas Kecil 3

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma pada Semester 4
Tahun Akademik 2024/2025

Disusun oleh:
Sebastian Enrico Nathanael (13523134)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

Daftar Isi

BAB I
DESKRIPSI TUGAS

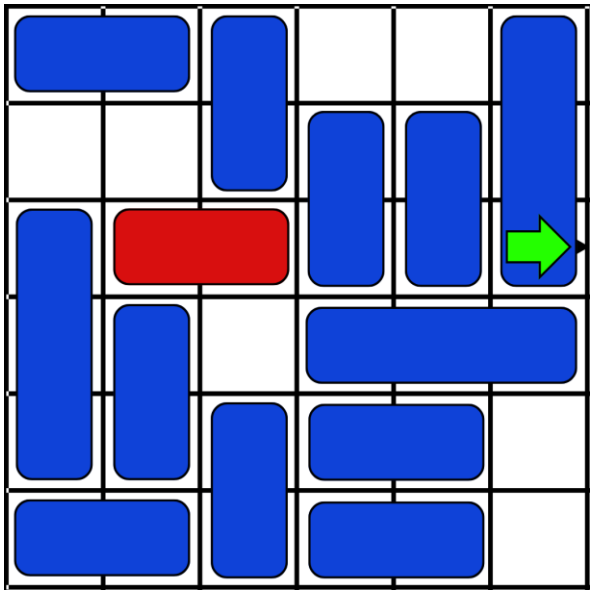
Gambar 1. Rush Hour Puzzle
(Sumber: https://www.thinkfun.com/en-US/products/educational-games/rush-hour-76582)

Rush Hour adalah sebuah permainan puzzle logika berbasis grid yang menantang pemain untuk menggeser kendaraan di dalam sebuah kotak (biasanya berukuran 6x6) agar mobil utama (biasanya berwarna merah) dapat keluar dari kemacetan melalui pintu keluar di sisi papan. Setiap kendaraan hanya bisa bergerak lurus ke depan atau ke belakang sesuai dengan orientasinya (horizontal atau vertikal), dan tidak dapat berputar. Tujuan utama dari permainan ini adalah memindahkan mobil merah ke pintu keluar dengan jumlah langkah seminimal mungkin.

Komponen penting dari permainan Rush Hour terdiri dari:

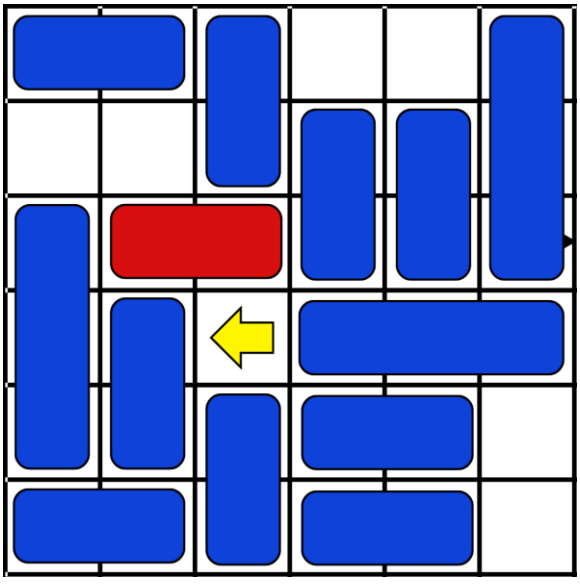
1. Papan – Papan merupakan tempat permainan dimainkan. Papan terdiri atas cell, yaitu sebuah singular point dari papan. Sebuah piece akan menempati cell-cell pada papan. Ketika permainan dimulai, semua piece telah diletakkan di dalam papan dengan konfigurasi tertentu berupa lokasi piece dan orientasi, antara horizontal atau vertikal. Hanya primary piece yang dapat digerakkan keluar papan melewati pintu keluar. Piece yang bukan primary piece tidak dapat digerakkan keluar papan. Papan memiliki satu pintu keluar yang pasti berada di dinding papan dan sejajar dengan orientasi primary piece.
2. Piece – Piece adalah sebuah kendaraan di dalam papan. Setiap piece memiliki posisi, ukuran, dan orientasi. Orientasi sebuah piece hanya dapat berupa horizontal atau vertikal– tidak mungkin diagonal. Piece dapat memiliki beragam ukuran, yaitu jumlah cell yang ditempati oleh piece. Secara standar, variasi ukuran sebuah piece adalah 2-piece (menempati 2 cell) atau 3-piece (menempati 3 cell). Suatu piece tidak dapat digerakkan melewati/menembus piece yang lain.
3. Primary Piece – Primary piece adalah kendaraan utama yang harus dikeluarkan dari papan (biasanya berwarna merah). Hanya boleh terdapat satu primary piece.
4. Pintu Keluar – Pintu keluar adalah tempat primary piece dapat digerakkan keluar untuk menyelesaikan permainan
5. Gerakan — Gerakan yang dimaksudkan adalah pergeseran piece di dalam permainan. Piece hanya dapat bergerak/bergeser lurus sesuai orientasinya (atas-bawah jika vertikal dan kiri-kanan jika horizontal). Suatu piece tidak dapat digerakkan melewati/menembus piece yang lain.

Ilustrasi kasus :
Diberikan sebuah papan berukuran 6 x 6 dengan 12 piece kendaraan dengan 1 piece merupakan primary piece. Piece ditempatkan pada papan dengan posisi dan orientasi sebagai berikut.

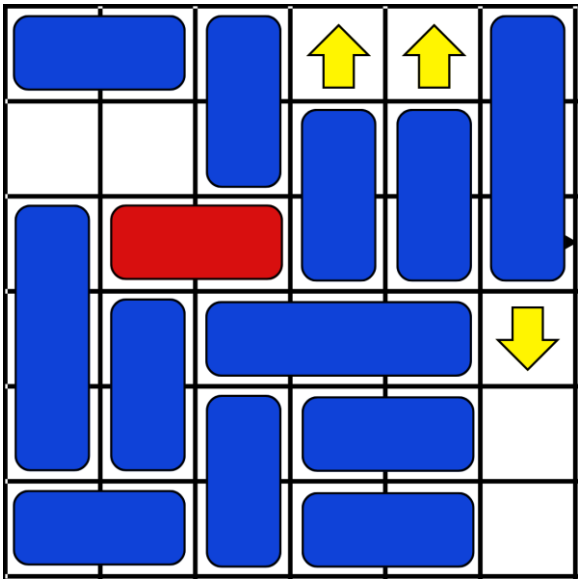


Gambar 2. Awal Permainan Game Rush Hour

Pemain dapat menggeser-geser piece (termasuk primary piece) untuk membentuk jalan lurus antara primary piece dan pintu keluar.

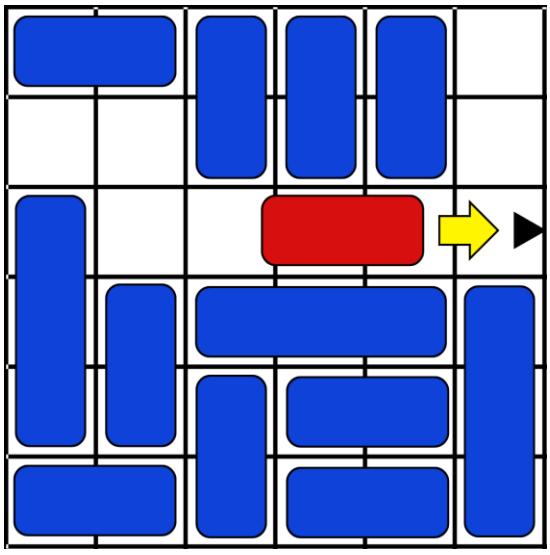


Gambar 3. Gerakan Pertama Game Rush Hour



Gambar 4. Gerakan Kedua Game Rush Hour

Puzzle berikut dinyatakan telah selesai apabila primary piece dapat digeser keluar papan melalui pintu keluar.



Gambar 5. Pemain Menyelesaikan Permainan

Agar lebih jelas, silahkan amati video cara bermain berikut: The New Rush Hour by ThinkFun!

Anda juga dapat melihat gif berikut untuk melihat contoh permainan Rush Hour Solution.

Spesifikasi Tugas Kecil 3:

● Buatlah program sederhana dalam bahasa C/C++/Java/Javascript yang mengimplementasikan algoritma pathfinding Greedy Best First Search, UCS (Uniform Cost Search), dan A* dalam menyelesaikan permainan Rush Hour.

● Tugas dapat dikerjakan individu atau berkelompok dengan anggota maksimal 2 orang (sangat disarankan). Boleh lintas kelas dan lintas kampus, tetapi tidak boleh sama dengan anggota kelompok pada tugas kecil Strategi Algoritma sebelumnya.

● Algoritma pathfinding minimal menggunakan satu heuristic (2 atau lebih jika mengerjakan bonus) yang ditentukan sendiri. Jika mengerjakan bonus, heuristic yang digunakan ditentukan berdasarkan input pengguna.

● Algoritma dijalankan secara terpisah. Algoritma yang digunakan ditentukan berdasarkan Input pengguna.

● Alur Program:

1. [INPUT] konfigurasi permainan/test case dalam format ekstensi .txt. File test case tersebut berisi:
    1. Dimensi Papan terdiri atas dua buah variabel A dan B yang membentuk papan berdimensi AxB
    2. Banyak piece BUKAN primary piece direpresentasikan oleh variabel integer N.
    3. Konfigurasi papan yang mencakup penempatan piece dan primary piece, serta lokasi pintu keluar. Primary Piece dilambangkan dengan huruf P dan pintu keluar dilambangkan dengan huruf K. Piece dilambangkan dengan huruf dan karakter selain P dan K, dan huruf/karakter berbeda melambangkan piece yang berbeda. Cell kosong dilambangkan dengan karakter ‘.’ (titik). (Catatan: ingat bahwa pintu keluar pasti berada di dinding papan dan sejajar dengan orientasi primary piece)

File .txt yang akan dibaca memiliki format sebagai berikut:

```
A B
N
konfigurasi_papan
```

Contoh Input

```
6 6
12
```

```
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.
```

keterangan: “K” adalah pintu keluar, “P” adalah primary piece, Titik (“.”) adalah cell kosong.
Contoh konfigurasi papan lain yang mungkin berdasarkan letak pintu keluar (X adalah piece/cell random)

K XXX XXX XXX	XXX KXXX XXX	XXX XXX XXX K

2. [INPUT] algoritma pathfinding yang digunakan
3. [INPUT] heuristic yang digunakan (bonus)
4. [OUTPUT] Banyaknya gerakan yang diperiksa (alias banyak ‘node’ yang dikunjungi)
5. [OUTPUT] Waktu eksekusi program
6. [OUTPUT] konfigurasi papan pada setiap tahap pergerakan/pergeseran. Output ini tidak harus diimplementasi apabila mengerjakan bonus output GUI. Gunakan print berwarna untuk menunjukkan pergerakan piece dengan jelas. Cukup mewarnakan primary piece, pintu keluar, dan piece yang digerakkan saja (boleh dengan highlight atau text color). Pastikan ketiga komponen tersebut memiliki warna berbeda.

Format sekuens adalah sebagai berikut:

```
Papan Awal
[konfigurasi_papan_awal]

Gerakan 1: [piece]-[arah gerak]
[konfigurasi_papan_gerakan_1]

Gerakan 2: [piece]-[arah gerak] [konfigurasi_papan_gerakan_2]
Gerakan [N]: [piece]-[arah gerak] [konfigurasi_papan_gerakan_N]
dst
```

Contoh Output

Papan Awal
AAB..F
..BCDF
GPPCDFK

GH.III
GHJ...
LLJMM.

Gerakan 1: I-kiri AAB..F
..BCDF
GPPCDFK GHIII.
GHJ...
LLJMM.

Gerakan 2: F-bawah
AAB…
..BCDF
GPPCDFK
GHIIIF
GHJ...
LLJMM.
dst

Keterangan: hanya sebagai contoh. Pastikan output jelas dan mudah dimengerti. Warna dan highlight hanya untuk menunjukan perubahan.

7. [OUTPUT] animasi gerakan-gerakan untuk mencapai solusi (bonus GUI).

Berkas laporan yang dikumpulkan adalah laporan dalam bentuk PDF yang setidaknya berisi:

Penjelasan algoritma UCS, Greedy Best First Search, dan A* (dan algoritma alternatif apabila mengerjakan bonus), bukan hanya berisi notasi pseudocode dan BUKAN IMPLEMENTASINYA melainkan algoritmanya.
2. Analisis algoritma UCS, Greedy Best First Search, dan A* (dan algoritma alternatif apabila mengerjakan bonus). Analisis minimal memuat jawaban dari pertanyaan-pertanyaan berikut:
1. Definisi dari f(n) dan g(n), sesuai dengan salindia kuliah.
2. Apakah heuristik yang digunakan pada algoritma A* admissible? Jelaskan sesuai definisi admissible dari salindia kuliah.
3. Pada penyelesaian Rush Hour, apakah algoritma UCS sama dengan BFS? (dalam artian urutan node yang dibangkitkan dan path yang dihasilkan sama)

4. Secara teoritis, apakah algoritma A* lebih efisien dibandingkan dengan algoritma UCS pada penyelesaian Rush Hour?
5. Secara teoritis, apakah algoritma Greedy Best First Search menjamin solusi optimal untuk penyelesaian Rush Hour?
3. Source program dalam bahasa pemrograman yang dipilih (pastikan bahwa program telah dapat dijalankan).
4. Tangkapan layar yang memperlihatkan input dan output (minimal sebanyak 4 buah contoh untuk masing-masing algoritma). Disarankan mencangkup semua kasus unik.
5. Hasil analisis percobaan algoritma pathfinding. Analisis dilakukan dalam bentuk paragraf/poin dan minimal memuat mengenai analisis kompleksitas algoritma program yang telah dikembangkan.
6. Penjelasan mengenai implementasi bonus jika mengerjakan.
7. Pranala ke repository yang berisi kode program.
● BONUS:
Pastikan sudah mengerjakan spesifikasi wajib sebelum mengerjakan bonus:
1. Implementasikan Algoritma Alternatif
Tambahkan minimal 1 (satu) implementasi algoritma pathfinding lain selain Greedy Best First Search, UCS, atau A*. Algoritma pathfinding alternatif tidak boleh berupa BFS atau DFS.
2. Implementasi Heuristic Alternatif Implementasikan 2 (dua) atau lebih heuristic (alias 1 atau lebih heuristic tambahan) yang dapat digunakan algoritma pathfinding dalam program kalian. Tambahkan input untuk memasukkan pilihan heuristic yang akan digunakan algoritma.

Graphical User Interface
Buatlah GUI untuk program yang Anda buat. Interface ini harus dapat menerima input secara graphical. Interface juga harus dapat mengeluarkan output berupa animasi gerakan-gerakan dari awal permainan sampai mencapai solusi. Kakas untuk implementasi GUI dibebaskan.
● Program disimpan dalam repository yang bernama Tucil3_NIM jika mengerjakan secara individu atau Tucil3_NIM1_NIM2 jika dikerjakan berkelompok. Berikut merupakan struktur dari isi repository tersebut:
1. Folder src berisi source code program.
2. Folder bin berisi executable file (Sesuaikan dengan bahasa pemrograman yang digunakan).
3. Folder test berisi solusi jawaban dari data uji yang digunakan dalam laporan.
4. Folder doc berisi laporan tugas kecil dalam bentuk PDF.
5. README yang minimal berisi:
a. Penjelasan singkat program yang dibuat.
b. Requirement program dan instalasi tertentu bila ada.
c. Cara mengkompilasi program bila perlu dikompilasi (pastikan dengan langkah yang jelas dan benar).
d. Cara menjalankan dan menggunakan program (pastikan dengan langkah yang jelas dan benar).
e. Author / identitas pembuat.

● Laporan dikumpulkan hari Rabu, 21 Mei 2025 pada alamat Google Form berikut paling lambat pukul 13.00 WIB:
https://forms.gle/bVRWcFhVssRjCzNx5
● Pertanyaan terkait tugas kecil ini bisa disampaikan melalui QnA berikut: https://bit.ly/QnA-Stima-25

BAB II
LANDASAN TEORI

2.1 UCS

UCS adalah algoritma pencarian yang menjelajahi simpul berdasarkan total biaya dari simpul awal ke simpul saat ini, atau fungsi g(n). UCS tidak mempertimbangkan estimasi ke tujuan sehingga sering disebut sebagai uninformed search. UCS menggunakan antrian prioritas berdasarkan g(n). UCS memilih node yang memiliki **biaya kumulatif terendah** dari simpul awal ke simpul tersebut. Algoritma ini menjamin solusi **optimal** jika semua biaya lintasan adalah positif.

UCS menggunakan struktur data **priority queue**, di mana elemen dengan total biaya terendah memiliki prioritas tertinggi. Proses pencarian dilakukan dengan mengekspansi node berdasarkan biaya yang paling murah hingga menemukan node tujuan.

g(n) adalah biaya kumulatif dari simpul awal ke simpul n, yaitu banyaknya langkah yang telah ditempuh.

2.2. Greedy Best First Search (GBFS)

Greedy Best First Search (GBFS) adalah algoritma pencarian yang termasuk dalam *informed search*, karena menggunakan fungsi heuristik untuk mengarahkan proses pencarian. Pada setiap langkah, algoritma ini akan memilih node dengan nilai heuristik *h(n)* paling kecil, yang mengindikasikan bahwa node tersebut dianggap paling "dekat" dengan tujuan.

Berbeda dari UCS yang mempertimbangkan biaya total dari awal, GBFS hanya memperhatikan estimasi jarak dari node saat ini ke tujuan, sehingga bersifat "rakus" terhadap jalan tercepat menurut estimasi tersebut.

2.3. A*

A* Search adalah algoritma pencarian informatif yang menggabungkan keunggulan dari UCS dan GBFS. Algoritma ini menggunakan fungsi evaluasi f(n) = g(n) + h(n), di mana:

- *g(n)* adalah biaya aktual dari simpul awal ke node *n*,
- *h(n)* adalah estimasi biaya dari *n* ke node tujuan (fungsi heuristik).

A* lebih efisien dibanding UCS dan lebih optimal dibanding GBFS, selama heuristik yang digunakan adalah *admissible* dan *consistent*.

Dengan menggabungkan biaya aktual dan estimasi, A* mampu menjamin solusi optimal jika heuristik yang digunakan bersifat admissible (tidak melebih-lebihkan biaya ke tujuan) dan consistent (memenuhi segitiga ketat).

BAB III
IMPLEMENTASI DAN PENGUJIAN

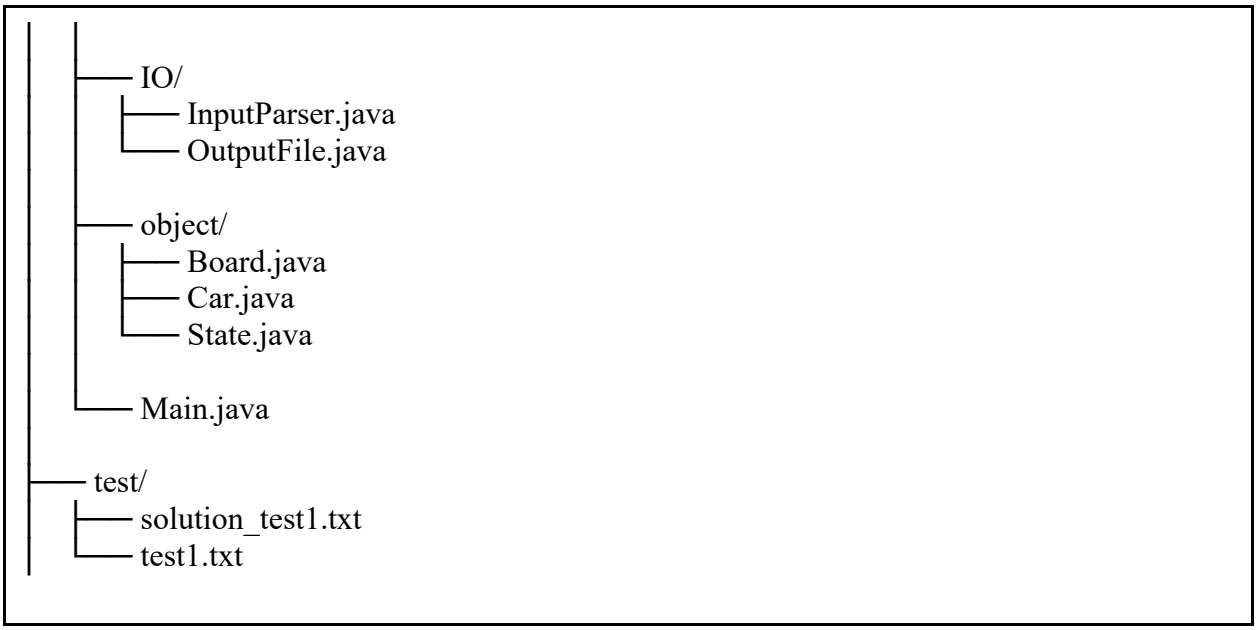
3.1 Implementasi dan Analisis Algoritma

3.1.1 UCS

3.2 Source Program

Struktur Folder dan File

```
Tucil3_13523134/
│
├── bin/
├── doc/
├── src/
│   ├── algorithm/
│   │   ├── AStar.java
│   │   ├── GreedyBestFirst.java
│   │   └── UCS.java
```



Algorithm Package

3.2.1. UCS.java

Source code

```java
package algorithm;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Set;

import object.Board;
import object.State;

public class UCS {
    private int nodesVisited;
    private long executionTime;
    private State initialState;
    private State goalState;

    public UCS(Board initialBoard) {
        this.initialState = new State(initialBoard);
        this.nodesVisited = 0;
```

```
        this.executionTime = 0;
        this.goalState = null;
    }

    public boolean execute() {
        long startTime = System.currentTimeMillis();

        // Priority queue sorted by cost (Uniform Cost Search uses g(n) only)
        PriorityQueue<State> openSet = new
PriorityQueue<>(Comparator.comparingInt(State::getCost));
        Set<Board> closedSet = new HashSet<>();

        openSet.add(initialState);

        while (!openSet.isEmpty()) {
            // Get the state with the lowest cost
            State current = openSet.poll();
            nodesVisited++;

            // Check if goal state is reached
            if (current.isGoal()) {
                goalState = current;
                executionTime = System.currentTimeMillis() - startTime;
                return true;
            }

            // Add current board to closed set
            closedSet.add(current.getBoard());

            // Generate and process next states
            for (State nextState : current.generateNextStates()) {
                // Skip if we've already visited this board configuration
                if (closedSet.contains(nextState.getBoard())) {
                    continue;
                }

                // Add to open set
                openSet.add(nextState);
            }
        }

        executionTime = System.currentTimeMillis() - startTime;
        return false;
    }

    public List<State> getSolutionPath() {
        if (goalState == null) {
            return Collections.emptyList();
        }
```

```
        List<State> path = new ArrayList<>();
        State current = goalState;

        // Trace back from goal to initial state
        while (current != null) {
            path.add(current);
            current = current.getParent();
        }

        // Reverse to get path from initial to goal
        Collections.reverse(path);
        return path;
    }

    public int getNodesVisited() {
        return nodesVisited;
    }

    public long getExecutionTime() {
        return executionTime;
    }

    public int getNumberOfMoves() {
        return goalState != null ? goalState.getCost() : -1;
    }
}
```

3.2.2. GreedyBestFirst.java

Source code

```
package algorithm;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Set;
import java.util.function.Function;
```

```java
import object.Board;
import object.Car;
import object.State;

public class GreedyBestFirst {
    private int nodesVisited;
    private long executionTime;
    private State initialState;
    private State goalState;
    private Function<Board, Integer> heuristicFunction;

    public GreedyBestFirst(Board initialBoard, Function<Board, Integer>
heuristicFunction) {
        this.initialState = new State(initialBoard);
        this.nodesVisited = 0;
        this.executionTime = 0;
        this.goalState = null;
        this.heuristicFunction = heuristicFunction;

        // Calculate heuristic for initial state
        initialState.setHeuristic(heuristicFunction.apply(initialBoard));
    }

    public GreedyBestFirst(Board initialBoard) {
        this(initialBoard, GreedyBestFirst::blockingCarsHeuristic);
    }

    public boolean execute() {
        long startTime = System.currentTimeMillis();

        // Priority queue sorted by heuristic value only for Greedy Best First Search
        PriorityQueue<State> openSet = new
PriorityQueue<>(Comparator.comparingInt(State::getHeuristic));
        Set<Board> closedSet = new HashSet<>();

        openSet.add(initialState);

        while (!openSet.isEmpty()) {
            // Get the state with the lowest heuristic value
            State current = openSet.poll();
            nodesVisited++;

            // Check if goal state is reached
            if (current.isGoal()) {
                goalState = current;
                executionTime = System.currentTimeMillis() - startTime;
                return true;
            }
```

```
            // Add current board to closed set
            closedSet.add(current.getBoard());

            // Generate and process next states
            for (State nextState : current.generateNextStates()) {
                // Skip if we've already visited this board configuration
                if (closedSet.contains(nextState.getBoard())) {
                    continue;
                }

                // Calculate heuristic for next state
                nextState.setHeuristic(heuristicFunction.apply(nextState.getBoard()));

                // Add to open set
                openSet.add(nextState);
            }
        }

        executionTime = System.currentTimeMillis() - startTime;
        return false;
    }

    public List<State> getSolutionPath() {
        if (goalState == null) {
            return Collections.emptyList();
        }

        List<State> path = new ArrayList<>();
        State current = goalState;

        // Trace back from goal to initial state
        while (current != null) {
            path.add(current);
            current = current.getParent();
        }

        // Reverse to get path from initial to goal
        Collections.reverse(path);
        return path;
    }

    public static int blockingCarsHeuristic(Board board) {
        Car primaryCar = board.getPrimaryCar();
        if (primaryCar == null) {
            return Integer.MAX_VALUE; // No primary car found
        }

        int blockingCars = 0;
        char[][] grid = board.getGrid();
```

```
        // Determine exit direction
        int exitRow = board.getExitRow();
        int exitCol = board.getExitCol();

        if (primaryCar.isHorizontal()) {
            // Primary car is horizontal, check cars blocking to the right or left
            int startRow = primaryCar.getRow();

            if (exitCol > primaryCar.getCol()) {
                // Exit is to the right
                int startCol = primaryCar.getCol() + primaryCar.getLength();
                for (int c = startCol; c <= exitCol; c++) {
                    if (c < board.getCols() && grid[startRow][c] != '.' &&
grid[startRow][c] != 'K') {
                        blockingCars++;
                    }
                }
            } else {
                // Exit is to the left
                for (int c = primaryCar.getCol() - 1; c >= exitCol; c--) {
                    if (c >= 0 && grid[startRow][c] != '.' && grid[startRow][c] != 'K') {
                        blockingCars++;
                    }
                }
            }
        } else {
            // Primary car is vertical, check cars blocking up or down
            int startCol = primaryCar.getCol();

            if (exitRow > primaryCar.getRow()) {
                // Exit is below
                int startRow = primaryCar.getRow() + primaryCar.getLength();
                for (int r = startRow; r <= exitRow; r++) {
                    if (r < board.getRows() && grid[r][startCol] != '.' &&
grid[r][startCol] != 'K') {
                        blockingCars++;
                    }
                }
            } else {
                // Exit is above
                for (int r = primaryCar.getRow() - 1; r >= exitRow; r--) {
                    if (r >= 0 && grid[r][startCol] != '.' && grid[r][startCol] != 'K') {
                        blockingCars++;
                    }
                }
            }
        }
```

```
        return blockingCars;
    }

    public static int distanceAndBlockingHeuristic(Board board) {
        Car primaryCar = board.getPrimaryCar();
        if (primaryCar == null) {
            return Integer.MAX_VALUE; // No primary car found
        }

        int blockingCars = blockingCarsHeuristic(board);
        int distance = 0;

        // Calculate distance from primary car to exit
        if (primaryCar.isHorizontal()) {
            int exitCol = board.getExitCol();
            if (exitCol > primaryCar.getCol()) {
                // Exit is to the right
                distance = exitCol - (primaryCar.getCol() + primaryCar.getLength() - 1);
            } else {
                // Exit is to the left
                distance = primaryCar.getCol() - exitCol;
            }
        } else {
            int exitRow = board.getExitRow();
            if (exitRow > primaryCar.getRow()) {
                // Exit is below
                distance = exitRow - (primaryCar.getRow() + primaryCar.getLength() - 1);
            } else {
                // Exit is above
                distance = primaryCar.getRow() - exitRow;
            }
        }

        // Combine distance and blocking cars (weighted)
        return distance + 2 * blockingCars;
    }

    public int getNodesVisited() {
        return nodesVisited;
    }

    public long getExecutionTime() {
        return executionTime;
    }

    public int getNumberOfMoves() {
        return goalState != null ? goalState.getCost() : -1;
    }
}
```

3.2.3 AStar.java

Source code

```java
package algorithm;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;
import java.util.Set;
import java.util.function.Function;

import object.Board;
import object.State;

public class AStar {
    private int nodesVisited;
    private long executionTime;
    private State initialState;
    private State goalState;
    private Function<Board, Integer> heuristicFunction;

    public AStar(Board initialBoard, Function<Board, Integer> heuristicFunction) {
        this.initialState = new State(initialBoard);
        this.nodesVisited = 0;
        this.executionTime = 0;
        this.goalState = null;
        this.heuristicFunction = heuristicFunction;

        initialState.setHeuristic(heuristicFunction.apply(initialBoard));
    }

    public AStar(Board initialBoard) {
        this(initialBoard, GreedyBestFirst::blockingCarsHeuristic);
    }

    public boolean execute() {
        long startTime = System.currentTimeMillis();

        PriorityQueue<State> openSet = new PriorityQueue<>();
        Map<Board, Integer> bestCostSoFar = new HashMap<>();
```

```
        Set<Board> closedSet = new HashSet<>();

        openSet.add(initialState);
        bestCostSoFar.put(initialState.getBoard(), 0);

        while (!openSet.isEmpty()) {
            State current = openSet.poll();
            nodesVisited++;

            // Check if goal state is reached
            if (current.isGoal()) {
                goalState = current;
                executionTime = System.currentTimeMillis() - startTime;
                return true;
            }

            // Skip if we've processed this state with a better cost
            Integer bestCost = bestCostSoFar.get(current.getBoard());
            if (bestCost != null && current.getCost() > bestCost) {
                continue;
            }

            // Add current board to closed set
            closedSet.add(current.getBoard());

            // Generate and process next states
            for (State nextState : current.generateNextStates()) {
                Board nextBoard = nextState.getBoard();

                // Skip if we've already visited this board configuration with a better
cost
                if (closedSet.contains(nextBoard)) {
                    continue;
                }

                nextState.setHeuristic(heuristicFunction.apply(nextBoard));

                // Check if we found a better path to this board
                Integer currentBestCost = bestCostSoFar.get(nextBoard);
                if (currentBestCost == null || nextState.getCost() < currentBestCost) {
                    bestCostSoFar.put(nextBoard, nextState.getCost());
                    openSet.add(nextState);
                }
            }
        }

        executionTime = System.currentTimeMillis() - startTime;
        return false;
    }
```

```
    public List<State> getSolutionPath() {
        if (goalState == null) {
            return Collections.emptyList();
        }

        List<State> path = new ArrayList<>();
        State current = goalState;

        // Trace back from goal to initial state
        while (current != null) {
            path.add(current);
            current = current.getParent();
        }

        // Reverse to get path from initial to goal
        Collections.reverse(path);
        return path;
    }

    public int getNodesVisited() {
        return nodesVisited;
    }

    public long getExecutionTime() {
        return executionTime;
    }

    public int getNumberOfMoves() {
        return goalState != null ? goalState.getCost() : -1;
    }
}
```

3.2.4. Car.java

Source code

```
package object;

public class Car {
    public char id;
    public int row, col;
    public int length;
    public boolean isHorizontal;
    public boolean isPrimary;
```

```
    public Car(char id, int row, int col, int length, boolean isHorizontal, boolean
isPrimary) {
        this.id = id;
        this.row = row;
        this.col = col;
        this.length = length;
        this.isHorizontal = isHorizontal;
        this.isPrimary = isPrimary;
    }

    public Car(Car other){
        this.id = other.id;
        this.row = other.row;
        this.col = other.col;
        this.length = other.length;
        this.isHorizontal = other.isHorizontal;
        this.isPrimary = other.isPrimary;
    }

    public char getId() {
        return id;
    }

    public int getRow() {
        return row;
    }

    public int getCol() {
        return col;
    }

    public int getLength() {
        return length;
    }

    public boolean isHorizontal() {
        return isHorizontal;
    }

    public boolean isPrimary() {
        return isPrimary;
    }

    public void setRow(int row) {
        this.row = row;
    }

    public void setCol(int col) {
        this.col = col;
```

```
    }

    public int canMoveRight(char[][] board){
        int steps = 0;
        for (int i = col + length; i < board[0].length; i++){
            if (board[row][i] == '.' || board[row][i] == id){
                steps++;
            } else {
                break;
            }
        }
        return steps;
    }

    public int canMoveLeft(char[][] board){
        int steps = 0;
        for (int i = col - 1; i >= 0; i++){
            if (board[row][i] == '.' || board[row][i] == id){
                steps++;
            } else {
                break;
            }
        }
        return steps;
    }

    public int canMoveUp(char[][] board){
        int steps = 0;
        for (int i = row - 1; i >= 0; i--){
            if (board[i][col] == '.' || board[i][col] == id){
                steps++;
            } else {
                break;
            }
        }
        return steps;
    }

    public int canMoveDown(char[][] board){
        int steps = 0;
        for (int i = row + length; i < board.length; i++){
            if (board[i][col] == '.' || board[i][col] == id){
                steps++;
            } else {
                break;
            }
        }
        return steps;
    }
```

```
    public Car moveRight(int steps) {
        return new Car(id, row, col + steps, length, isHorizontal, isPrimary);
    }

    public Car moveLeft(int steps) {
        return new Car(id, row, col - steps,  length, isHorizontal, isPrimary);
    }

    public Car moveUp(int steps) {
        return new Car(id, row - steps, col,  length, isHorizontal, isPrimary);
    }

    public Car moveDown(int steps) {
        return new Car(id, row + steps, col, length, isHorizontal, isPrimary);
    }

    public Car clone() {
        return new Car(id, row, col, length, isHorizontal, isPrimary);
    }

    // @Override
    // public boolean equals(Object o) {
    //     if (!(o instanceof Position)) return false;
    //     return this.row == row && this.col == col;
    // }

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

    public Car move(int newRow, int newCol) {
        return new Car(this.id, newRow, newCol, this.length, this.isHorizontal,
this.isPrimary);
    }

    public int[][] getOccupiedCells() {
        int[][] cells = new int[length][2];

        for (int i = 0; i < length; i++) {
            if (isHorizontal) {
                cells[i][0] = row;
                cells[i][1] = col + i;
            } else {
                cells[i][0] = row + i;
                cells[i][1] = col;
            }
        }
```

```
        return cells;
    }


    @Override
    public String toString() {
        return "Car [id=" + id + ", position=(" + row + "," + col + "), length=" + length
+
                ", orientation=" + (isHorizontal ? "horizontal" : "vertical") +
                ", isPrimary=" + isPrimary + "]";
    }

@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;

    Car other = (Car) obj;
    return id == other.id &&
           row == other.row &&
           col == other.col &&
           length == other.length &&
           isHorizontal == other.isHorizontal &&
           isPrimary == other.isPrimary;
}

@Override
public int hashCode() {
    int result = Character.hashCode(id);
    result = 31 * result + row;
    result = 31 * result + col;
    result = 31 * result + length;
    result = 31 * result + Boolean.hashCode(isHorizontal);
    result = 31 * result + Boolean.hashCode(isPrimary);
    return result;
}



}
```

3.2.5. Board.java

Source code

```
package object;

import java.util.ArrayList;
```

```java
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Board {
    public int rows;
    public int cols;
    public char[][] grid;
    public Map<Character, Car> cars;
    public int exitRow;
    public int exitCol;

    public Board(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        this.grid = new char[rows][cols];
        this.cars = new HashMap<>();

        // Initialize grid with empty cells
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                grid[r][c] = '.';
            }
        }
    }

    public Board copy() {
        Board newBoard = new Board(rows, cols);
        newBoard.exitRow = this.exitRow;
        newBoard.exitCol = this.exitCol;

        // Copy grid
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                newBoard.grid[r][c] = this.grid[r][c];
            }
        }

        // Copy cars
        for (Car car : this.cars.values()) {
            newBoard.addCar(car);
        }

        return newBoard;
    }

    public void addCar(Car car) {
        cars.put(car.getId(), car);
```

```
        // Update grid with car positions
        int[][] cells = car.getOccupiedCells();
        for (int[] cell : cells) {
            int r = cell[0];
            int c = cell[1];
            if (r >= 0 && r < rows && c >= 0 && c < cols && grid[r][c] != 'K') {
                grid[r][c] = car.getId();
            }
        }
    }

    public void setExit(int row, int col) {
        this.exitRow = row;
        this.exitCol = col;
        if (row >= 0 && row < rows && col >= 0 && col < cols) {
            grid[row][col] = 'K'; // K represents the exit
        }
    }

    public Board moveCar(char carId, int newRow, int newCol) {
        Car car = cars.get(carId);
        if (car == null) {
            return null; // Car not found
        }

        // debug cetak tujuan gerak
        // System.out.println("Mencoba memindahkan mobil " + carId + " ke posisi (" +
newRow + "," + newCol + ")");

        // Check if the move is valid
        if (!isValidMove(car, newRow, newCol)) {
        //     System.out.println(">> Gerakan TIDAK VALID untuk mobil " + carId + " ke ("
+ newRow + "," + newCol + ")");
        // this.printBoard(); // debug tampilkan board sebelum gerakan
            return null;
        }

        // Create a new board with the updated car position
        Board newBoard = this.copy();

        // Remove old car positions
        int[][] oldCells = car.getOccupiedCells();
        // for (int[] cell : oldCells) {
        //     int r = cell[0];
        //     int c = cell[1];
        //     if (r >= 0 && r < rows && c >= 0 && c < cols && grid[r][c] == carId) {
        //         newBoard.grid[r][c] = '.';
        //     }
        // }
```

```
        for (int[] cell : oldCells) {
            int r = cell[0];
            int c = cell[1];
            if (r >= 0 && r < rows && c >= 0 && c < cols) {
            newBoard.grid[r][c] = '.';  // jangan dicek carId lagi, langsung kosongkan
            }
        }

        // Add car with new position
        Car movedCar = car.move(newRow, newCol);
        newBoard.cars.put(carId, movedCar);

        // Update grid with new car positions
        int[][] newCells = movedCar.getOccupiedCells();
        for (int[] cell : newCells) {
            int r = cell[0];
            int c = cell[1];
            if (r >= 0 && r < rows && c >= 0 && c < cols) {
                // Don't overwrite the exit marker
                if (!(r == exitRow && c == exitCol)) {
                    newBoard.grid[r][c] = carId;
                }
            }
        }

        return newBoard;
    }

    public void validatePrimaryCarExitAlignment() {
    Car primary = getPrimaryCar();
    if (primary == null) {
        throw new IllegalArgumentException("Tidak ditemukan primary piece (P) pada
papan.");
    }

    // Jika horizontal, exit harus di baris yang sama
    if (primary.isHorizontal()) {
        if (primary.getRow() != exitRow) {
            throw new IllegalArgumentException("Primary piece P horizontal tidak sejajar
dengan pintu keluar (K).");
        }
    } else {
        if (primary.getCol() != exitCol) {
            throw new IllegalArgumentException("Primary piece P vertikal tidak sejajar
dengan pintu keluar (K).");
        }
    }
}
```

```
    public boolean isValidMove(Car car, int newRow, int newCol) {
        // Check if the car is moving along its orientation
        if (car.isHorizontal() && newRow != car.getRow()) {
            return false; // Horizontal car must move horizontally
        }
        if (!car.isHorizontal() && newCol != car.getCol()) {
            return false; // Vertical car must move vertically
        }

        // Buat salinan grid tanpa mobil yang sedang dipindah
    char[][] tempGrid = new char[rows][cols];
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            tempGrid[r][c] = grid[r][c];
        }
    }

    // Hapus posisi mobil yang sedang diuji
    for (int[] cell : car.getOccupiedCells()) {
        int r = cell[0];
        int c = cell[1];
        if (r >= 0 && r < rows && c >= 0 && c < cols && tempGrid[r][c] == car.getId()) {
            tempGrid[r][c] = '.';  // kosongkan
        }
    }

    // Cek apakah posisi barunya valid
    for (int i = 0; i < car.getLength(); i++) {
        int r = car.isHorizontal() ? newRow : newRow + i;
        int c = car.isHorizontal() ? newCol + i : newCol;

        if (r < 0 || r >= rows || c < 0 || c >= cols) return false;
        if (tempGrid[r][c] != '.' && tempGrid[r][c] != 'K') return false;
    }
    return true;
}

    public List<BoardMove> generateMoves() {
        List<BoardMove> possibleMoves = new ArrayList<>();

        for (Car car : cars.values()) {
            int row = car.getRow();
            int col = car.getCol();

            if (car.isHorizontal()) {
                // Try moving left
                for (int newCol = col - 1; newCol >= 0; newCol--) {
                    Board newBoard = moveCar(car.getId(), row, newCol);
                    if (newBoard != null) {
```

```
                        possibleMoves.add(new BoardMove(newBoard, car.getId(), "left"));
                        break;
                    }

                }

                // Try moving right
                for (int newCol = col + 1; newCol <= cols - car.getLength(); newCol++) {
                    Board newBoard = moveCar(car.getId(), row, newCol);
                    if (newBoard != null) {
                        possibleMoves.add(new BoardMove(newBoard, car.getId(), "right"));
                        break;
                    }
                }
            } else {
                // Try moving up
                for (int newRow = row - 1; newRow >= 0; newRow--) {
                    Board newBoard = moveCar(car.getId(), newRow, col);
                    if (newBoard != null) {
                        possibleMoves.add(new BoardMove(newBoard, car.getId(), "up"));
                        break;
                    }
                }

                // Try moving down
                for (int newRow = row + 1; newRow <= rows - car.getLength(); newRow++) {
                    Board newBoard = moveCar(car.getId(), newRow, col);
                    if (newBoard != null) {
                        possibleMoves.add(new BoardMove(newBoard, car.getId(), "down"));
                        break;
                    }
                }
            }
            if (car.isPrimary() && car.isHorizontal() && exitRow == car.getRow()) {
                int carRight = car.getCol() + car.getLength() - 1;
                if (exitCol > carRight) {
                    boolean pathClear = true;
                for (int c = carRight + 1; c <= exitCol; c++) {
                    char ch = grid[exitRow][c];
                if (ch != '.' && ch != 'K') {
                    pathClear = false;
                    break;
                }
            }
            if (pathClear) {
                Board newBoard = moveCar(car.getId(), car.getRow(), exitCol - car.getLength()
+ 1);
                if (newBoard != null) {
                    possibleMoves.add(new BoardMove(newBoard, car.getId(), "right-to-exit"));
```

```
            }
        }
    }
}
        }

        return possibleMoves;
    }

    public boolean isSolved() {
        Car primaryCar = null;
        for (Car car : cars.values()) {
            if (car.isPrimary()) {
                primaryCar = car;
                break;
            }
        }

        if (primaryCar == null) {
            return false;
        }
        int carRow = primaryCar.getRow();
        int carCol = primaryCar.getCol();
        int carLength = primaryCar.getLength();

        // For horizontal primary car, check if it can reach the exit
        if (primaryCar.isHorizontal()) {
            // Exit must be on the same row as the primary car
            if (carRow != exitRow) {
                return false;
            }

            int carLeft = carCol;
            int carRight = carCol + carLength - 1;

            // Selesaikan jika ujung kanan berada tepat di exit (ke kanan)
            if (carRight == exitCol) return true;

        // Selesaikan jika ujung kiri berada tepat di exit (ke kiri)
            if (carLeft == exitCol) return true;
            // If exit is on the right edge
            if (exitCol == cols - 1) {
                // Check if the path to the exit is clear
                for (int c = carRight + 1; c < cols; c++) {
                    if (grid[exitRow][c] != '.' && grid[exitRow][c] != 'K') {
                        return false;
                    }
                }
                return carRight == exitCol;
```

```
            }
            // If exit is on the left edge
            else if (exitCol == 0) {
                // Check if the path to the exit is clear
                for (int c = primaryCar.getCol() - 1; c >= 0; c--) {
                    if (grid[exitRow][c] != '.' && grid[exitRow][c] != 'K') {
                        return false;
                    }
                }
                return carLeft == exitCol;
            }
        }
        // For vertical primary car, check if it can reach the exit
        else {
            // Exit must be on the same column as the primary car
            if (exitCol != primaryCar.getCol()) {
                return false;
            }

            int carBottomEdge = primaryCar.getRow() + primaryCar.getLength() - 1;

            // If exit is on the bottom edge
            if (exitRow == rows - 1) {
                // Check if the path to the exit is clear
                for (int r = carBottomEdge + 1; r < rows; r++) {
                    if (grid[r][exitCol] != '.' && grid[r][exitCol] != 'K') {
                        return false;
                    }
                }
                return true;
            }
            // If exit is on the top edge
            else if (exitRow == 0) {
                // Check if the path to the exit is clear
                for (int r = primaryCar.getRow() - 1; r >= 0; r--) {
                    if (grid[r][exitCol] != '.' && grid[r][exitCol] != 'K') {
                        return false;
                    }
                }
                return true;
            }
        }

        return false;
    }

    public Car getPrimaryCar() {
        for (Car car : cars.values()) {
            if (car.isPrimary()) {
```

```
                return car;
            }
        }
        return null;
    }

    public void printBoard() {
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            System.out.print(grid[r][c]);
        }
        System.out.println();
        }
    }

    public static class BoardMove {
        private Board board;
        private char carId;
        private String direction;

        public BoardMove(Board board, char carId, String direction) {
            this.board = board;
            this.carId = carId;
            this.direction = direction;
        }

        public Board getBoard() {
            return board;
        }

        public char getCarId() {
            return carId;
        }

        public String getDirection() {
            return direction;
        }

        @Override
        public String toString() {
            return carId + "-" + direction;
        }
    }

    // Getters
    public int getRows() {
        return rows;
    }
```

```
    public int getCols() {
        return cols;
    }

    public char[][] getGrid() {
        return grid;
    }

    public Map<Character, Car> getCars() {
        return cars;
    }

    public int getExitRow() {
        return exitRow;
    }

    public int getExitCol() {
        return exitCol;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                sb.append(grid[r][c]);
            }
            if (r < rows - 1) {
                sb.append("\n");
            }
        }
        return sb.toString();
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;

        Board other = (Board) obj;
        return this.cars.equals(other.cars);
    }


    @Override
    public int hashCode() {
        return cars.hashCode();
    }
}
```

3.2.6. State.java

Source code

```
package object;

import java.util.List;
import java.util.Objects;


public class State implements Comparable<State> {
    public Board board;             // Current board configuration
    public State parent;            // Parent state
    public Board.BoardMove move;    // Move that led to this state
    public int cost;                // Cost to reach this state (number of moves)
    public int heuristic;           // Heuristic value

    public State(Board board) {
        this.board = board;
        this.parent = null;
        this.move = null;
        this.cost = 0;
        this.heuristic = 0;
    }

    public State(Board board, State parent, Board.BoardMove move) {
        this.board = board;
        this.parent = parent;
        this.move = move;
        this.cost = parent.cost + 1; // Increment cost by 1 for each move
        this.heuristic = 0;          // To be calculated later
    }
    private int getNewRowFromDirection(Car car, String direction) {
    return switch (direction) {
        case "up" -> car.getRow() - 1;
        case "down" -> car.getRow() + 1;
        default -> car.getRow(); // horizontal move
    };
}

private int getNewColFromDirection(Car car, String direction) {
    return switch (direction) {
        case "left" -> car.getCol() - 1;
        case "right" -> car.getCol() + 1;
        default -> car.getCol(); // vertical move
    };
}

    // public List<State> generateNextStates() {
    //     List<Board.BoardMove> possibleMoves = board.generateMoves();
```

```
    //      return possibleMoves.stream()
    //      //          .map(move -> new State(move.getBoard(), this, move))
    //      //          .toList(); debug
    //      .peek(move -> {
    //      // System.out.println("→ Menambahkan gerakan: " + move.getCarId() + " ke " +
move.getDirection());
    //      move.getBoard().printBoard(); // pastikan tidak ada board null
    // })
    // .map(move -> new State(move.getBoard(), this, move))
    // .toList();
    // }
    public List<State> generateNextStates() {
    List<Board.BoardMove> possibleMoves = board.generateMoves();

    return possibleMoves.stream()
        .filter(move -> {
            // Validasi ekstra apakah move ini memang benar-benar valid
            Car car = board.getCars().get(move.getCarId());
            int newRow = car.isHorizontal() ? car.getRow() : getNewRowFromDirection(car,
move.getDirection());
            int newCol = car.isHorizontal() ? getNewColFromDirection(car,
move.getDirection()) : car.getCol();
            boolean valid = board.isValidMove(car, newRow, newCol);
            if (!valid) {
                // System.out.println("Gerakan invalid" di );
                // board.printBoard();
            }
            return valid;
        })
        .map(move -> new State(move.getBoard(), this, move))
        .toList();
}

    public boolean isGoal() {
        return board.isSolved();
    }

    public void setHeuristic(int heuristic) {
        this.heuristic = heuristic;
    }

    public int getFValue() {
        return cost + heuristic;
    }

    /**
     * Compare states based on their f(n) values
     */
    @Override
```

```
    public int compareTo(State other) {
        // Compare based on f(n) = g(n) + h(n)
        int fValueDiff = this.getFValue() - other.getFValue();
        if (fValueDiff != 0) {
            return fValueDiff;
        }

        // If f values are equal, prefer lower h values (more informed)
        return this.heuristic - other.heuristic;
    }

    // Getters
    public Board getBoard() {
        return board;
    }

    public State getParent() {
        return parent;
    }

    public Board.BoardMove getMove() {
        return move;
    }

    public int getCost() {
        return cost;
    }

    public int getHeuristic() {
        return heuristic;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;

        State other = (State) obj;
        return Objects.equals(board, other.board);
    }

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

    @Override
    public String toString() {
        return "State [cost=" + cost + ", heuristic=" + heuristic + ", f=" + getFValue()
```

```
+ "]";
    }
}
```

3.2.7. InputParser.java

Source code

```
package IO;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.InputMismatchException;
import java.util.List;
import java.util.Set;

import object.Board;
import object.Car;

public class InputParser {

public static Board parse(String filename) throws IOException, InputMismatchException,
FileNotFoundException {
    BufferedReader reader = new BufferedReader(new FileReader(filename));

    String[] dims = reader.readLine().trim().split("\\s+");
    int expectedRows = Integer.parseInt(dims[0]);
    int expectedCols = Integer.parseInt(dims[1]);

    int nonPrimaryCars = Integer.parseInt(reader.readLine().trim());

    // Baca semua sisa baris sebagai isi grid
    String line;
    List<String> lineList = new ArrayList<>();
    int maxCols = 0;
    while ((line = reader.readLine()) != null) {
        lineList.add(line);
        maxCols = Math.max(maxCols, line.length());
    }

    int rows = lineList.size();
    int cols = maxCols;

    // Buat grid
    char[][] grid = new char[rows][cols];
```

```
    for (int i = 0; i < rows; i++) {
        String l = lineList.get(i);
        for (int j = 0; j < cols; j++) {
            grid[i][j] = (j < l.length()) ? l.charAt(j) : ' ';
        }
    }

    Board board = new Board(rows, cols);
    int exitRow = -1, exitCol = -1;
    int exitCount = 0;

    Set<Character> visited = new HashSet<>();
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            char c = grid[i][j];

            if (c == 'K') {
                exitRow = i;
                exitCol = j;
                exitCount++;
                continue;
            }

            if (c == '.' || c == ' ' || visited.contains(c)) continue;

            visited.add(c);
            boolean horizontal = (j + 1 < cols && grid[i][j + 1] == c);
            int length = 1;

            if (horizontal) {
                while (j + length < cols && grid[i][j + length] == c) length++;
            } else {
                while (i + length < rows && grid[i + length][j] == c) length++;
            }

            boolean isPrimary = (c == 'P');
            Car car = new Car(c, i, j, length, horizontal, isPrimary);
            board.addCar(car);
        }
        if (exitCount > 1) {
            System.out.println("pintu keluar K tidak valid, ditemukan: " + exitCount);
            break;
        }
    }
    board.setExit(exitRow, exitCol);
    board.validatePrimaryCarExitAlignment();
    reader.close();
    return board;
}
```

```
}
```

3.2.8. OutputFile.java

Source code

```
package IO;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import object.Board;
import object.State;

/**
 * Formats and outputs the solution to the Rush Hour puzzle.
 */
public class OutputFile {

    // ANSI color codes for console output
    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 YELLOW = "\u001B[33m";
    public static final String BLUE = "\u001B[34m";

    /**
     * Print the solution path to the console with colored output
     *
     * @param path         Solution path
     * @param nodesVisited Number of nodes visited during search
     * @param time         Execution time in milliseconds
     */
    public static void printSolution(List<State> path, int nodesVisited, long time) {
        if (path.isEmpty()) {
            System.out.println("No solution found.");
            return;
        }

        System.out.println("Solution found!");
        System.out.println("Number of moves: " + (path.size() - 1));
        System.out.println("Nodes visited: " + nodesVisited);
        System.out.println("Execution time: " + time + " ms");
        System.out.println();
```

```
        // Print initial state
        System.out.println("Papan Awal");
        printBoardColored(path.get(0).getBoard(), null);

        // Print each step
        for (int i = 1; i < path.size(); i++) {
            State state = path.get(i);
            System.out.println("Gerakan " + i + ": " + state.getMove().toString());
            printBoardColored(state.getBoard(), state.getMove().getCarId());
        }
    }

    /**
     * Print a board with color highlighting
     *
     * @param board        Board to print
     * @param movedCarId   ID of the car that was moved (null for initial state)
     */
    private static void printBoardColored(Board board, Character movedCarId) {
        char[][] grid = board.getGrid();

        for (int r = 0; r < board.getRows(); r++) {
            for (int c = 0; c < board.getCols(); c++) {
                char cell = grid[r][c];

                if (cell == 'P') {
                    // Primary car in red
                    System.out.print(RED + cell + RESET);
                } else if (cell == 'K') {
                    // Exit in green
                    System.out.print(GREEN + cell + RESET);
                } else if (movedCarId != null && cell == movedCarId) {
                    // Moved car in yellow
                    System.out.print(YELLOW + cell + RESET);
                } else {
                    // Other cells
                    System.out.print(cell);
                }
            }
            System.out.println();
        }
        System.out.println();
    }

    /**
     * Save the solution to a file
     *
     * @param path         Solution path
```

```
     * @param nodesVisited Number of nodes visited during search
     * @param time         Execution time in milliseconds
     * @param filePath     Path to save the solution
     * @throws IOException If there's an error writing to the file
     */
    public static void saveSolution(List<State> path, int nodesVisited, long time, String
filePath) throws IOException {
        try (PrintWriter writer = new PrintWriter(new FileWriter(filePath))) {
            if (path.isEmpty()) {
                writer.println("No solution found.");
                return;
            }

            writer.println("Solution found!");
            writer.println("Number of moves: " + (path.size() - 1));
            writer.println("Nodes visited: " + nodesVisited);
            writer.println("Execution time: " + time + " ms");
            writer.println();

            // Print initial state
            writer.println("Papan Awal");
            writer.println(path.get(0).getBoard().toString());
            writer.println();

            // Print each step
            for (int i = 1; i < path.size(); i++) {
                State state = path.get(i);
                writer.println("Gerakan " + i + ": " + state.getMove().toString());
                writer.println(state.getBoard().toString());
                writer.println();
            }
        }
    }
}
```

3.2.9. Main.java

Spurce Code

```
import java.util.List;
import java.util.Scanner;
import java.util.function.Function;

import IO.InputParser;
import IO.OutputFile;
import algorithm.AStar;
import algorithm.GreedyBestFirst;
import algorithm.UCS;
import object.Board;
```

```
import object.State;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Masukkan path file input (tanpa .txt): ");
        String filename = scanner.nextLine().trim();
        String filepath = "../test/" + filename + ".txt";

        System.out.println("Pilih algoritma (UCS / GBFS / AStar): ");
        String algo = scanner.nextLine().trim().toLowerCase();

        Function<Board, Integer> heuristicFunction =
GreedyBestFirst::blockingCarsHeuristic;
        if (algo.equals("gbfs") || algo.equals("astar")) {
            System.out.println("Pilih heuristic (blocking / distance):");
            String heuristic = scanner.nextLine().trim().toLowerCase();
            System.out.println("\n");

            if (heuristic.equals("distance")) {
                heuristicFunction = GreedyBestFirst::distanceAndBlockingHeuristic;
            } else if (!heuristic.equals("blocking")) {
                System.out.println("Heuristic tidak dikenali. Menggunakan heuristic
default (blocking).");
            }
        }

        try {
            Board board = InputParser.parse(filepath);
            List<State> path = null;
            int nodesVisited = 0;
            long executionTime = 0;
            System.out.println("Papan Awal:");
            board.printBoard();
            System.out.println("Exit is at: " + board.getExitRow() + "," +
board.getExitCol());


            switch (algo) {
                case "ucs":
                    UCS ucs = new UCS(board);
                    if (ucs.execute()) {
                        path = ucs.getSolutionPath();
                        nodesVisited = ucs.getNodesVisited();
                        executionTime = ucs.getExecutionTime();
                    }
                    break;
                case "gbfs":
```

```
                    GreedyBestFirst gbfs = new GreedyBestFirst(board, heuristicFunction);
                    if (gbfs.execute()) {
                        path = gbfs.getSolutionPath();
                        nodesVisited = gbfs.getNodesVisited();
                        executionTime = gbfs.getExecutionTime();
                    }
                    break;
                case "astar":
                    AStar astar = new AStar(board, heuristicFunction);
                    if (astar.execute()) {
                        path = astar.getSolutionPath();
                        nodesVisited = astar.getNodesVisited();
                        executionTime = astar.getExecutionTime();
                    }
                    break;
                default:
                    System.out.println("Algoritma tidak dikenali.");
                    return;
            }

            if (path != null && !path.isEmpty()) {
                OutputFile.printSolution(path, nodesVisited, executionTime);
                OutputFile.saveSolution(path, nodesVisited, executionTime,
"../test/solution_" + filename + ".txt");
                System.out.println("Solusi ditulis ke file: solution_"+ filename +
".txt");
            } else {
                System.out.println("Tidak ditemukan solusi.");
            }

        } catch (Exception e) {
            System.err.println("Terjadi kesalahan saat menjalankan program:");
            e.printStackTrace();
        }

        scanner.close();
    }
}
```

3.3. Testing (Pengujian)

3.3.1. UCS Algorithm

Test Case 1

Input
6 6 12 AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.
Output
Solution found! Number of moves: 7 Nodes visited: 38496 Execution time: 1394 ms Papan Awal AAB..F. ..BCDF. GPPCDFK GH.III. GHJ.... LLJMM.. Gerakan 1: I-left AAB..F. ..BCDF. GPPCDFK GHIII.. GHJ.... LLJMM.. Gerakan 2: D-up AAB.DF. ..BCDF. GPPC.FK GHIII.. GHJ.... LLJMM.. Gerakan 3: C-up AABCDF. ..BCDF. GPP..FK

GHIII..
GHJ....
LLJMM..

Gerakan 4: F-down
AABCD..
..BCDF.
GPP..FK
GHIIIF.
GHJ....
LLJMM..

Gerakan 5: F-down
AABCD..
..BCD..
GPP..FK
GHIIIF.
GHJ..F.
LLJMM..

Gerakan 6: F-down
AABCD..
..BCD..
GPP...K
GHIIIF.
GHJ..F.
LLJMMF.

Gerakan 7: P-right-to-exit
AABCD..
..BCD..
G....PK
GHIIIF.
GHJ..F.
LLJMMF.

Test Case 2

Input
6 6 11 AAB..F G.BCDF KGPPCDF GHJIII

GHJ... LL.MM.
Output
No solution found

Test Case 3

Input
6 6 1 ..PP..K .B.... .B.... ....
Output
Solution found! Number of moves: 1 Nodes visited: 6 Execution time: 4 ms Papan Awal ..PP..K .B..... .B..... .... Gerakan 1: P-right-to-exit .....PK .B..... .B..... ....

Test Case 4

Input
6 6 7 AABCCD ..B..D ..BPPEK .....E ..FF.. ..GG..
Output
Solution found! Number of moves: 2 Nodes visited: 16 Execution time: 7 ms Papan Awal AABCCD. ..B..D. ..BPPEK .....E. ..FF... ..GG... Gerakan 1: E-down AABCCD. ..B..D. ..BPP.K .....E. ..FF.E. ..GG... Gerakan 2: P-right-to-exit AABCCD. ..B..D. ..B..PK .....E. ..FF.E. ..GG...

Test Case 5

Input

```
4 5
6
   K
FAAAA
F.BBP
F.EEP
SS.GG
```

Output

```
Solution found!
Number of moves: 3
Nodes visited: 67
Execution time: 11 ms

Papan Awal
....K
FAAAA
F.BBP
F.EEP
SS.GG
.....

Gerakan 1: S-right
....K
FAAAA
F.BBP
F.EEP
.SSGG
.....

Gerakan 2: F-down
....K
.AAAA
F.BBP
F.EEP
FSSGG
.....

Gerakan 3: A-left
....K
AAAA.
F.BBP
F.EEP
FSSGG
.....
```

Test Case 6

Input
6 6 9 .AABCC .DDBE. .FPPE.K .F.... ..GHHH IIG...
Output
Solution found! Number of moves: 3 Nodes visited: 279 Execution time: 28 ms Papan Awal .AABCC. .DDBE.. .FPPE.K .F..... ..GHHH. IIG.... Gerakan 1: C-right .AAB.CC .DDBE.. .FPPE.K .F..... ..GHHH. IIG.... Gerakan 2: E-up .AABECC .DDBE.. .FPP..K .F..... ..GHHH. IIG....

Gerakan 3: P-right-to-exit .AABECC .DDBE.. .F...PK .F..... ..GHHH. IIG....

Test Case 7

Input
10 10 15 AAB....F.. ..B.CCCF.M D.B..T..OM D..EET..OM D.PP.T..O.K D..LLL...H .....GGGH J...NNN..H JIII..... J.....
Output

Test Case 8

Input
7 7 9 KKK.... RRRRQ.. K..NQMPP W.NLLM. W.X..M. ..X.... ..OOO..
Output

pintu keluar K tidak valid, ditemukan: 3

Test Case 9

Input
5 5 6   K A.ABB A.CCC DDDF. ...FP .EEEP
Output
Solution found! Number of moves: 2 Nodes visited: 15 Execution time: 19 ms Papan Awal ....K A..BB A.CCC DDDF. ...FP .EEEP Gerakan 1: C-left ....K A..BB ACCC. DDDF. ...FP .EEEP Gerakan 2: B-left ....K A.BB. ACCC. DDDF. ...FP .EEEP

3.3.2. A* Algorithm

Test Case 1 blocking

Input
6 7 12 AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.
Output
Solution found! Number of moves: 7 Nodes visited: 227 Execution time: 31 ms Papan Awal AAB..F. ..BCDF. GPPCDFK GH.III. GHJ.... LLJMM.. Gerakan 1: C-up AABC.F. ..BCDF. GPP.DFK GH.III. GHJ.... LLJMM.. Gerakan 2: D-up AABCDF. ..BCDF. GPP..FK GH.III. GHJ.... LLJMM.. Gerakan 3: I-left AABCDF.

..BCDF.
GPP..FK
GHIII..
GHJ....
LLJMM..

Gerakan 4: F-down
AABCD..
..BCDF.
GPP..FK
GHIIIF.
GHJ....
LLJMM..

Gerakan 5: F-down
AABCD..
..BCD..
GPP..FK
GHIIIF.
GHJ..F.
LLJMM..

Gerakan 6: F-down
AABCD..
..BCD..
GPP...K
GHIIIF.
GHJ..F.
LLJMMF.

Gerakan 7: P-right-to-exit
AABCD..
..BCD..
G....PK
GHIIIF.
GHJ..F.
LLJMMF.

Test Case 2 blocking

Input
6 6 11

AAB..F G.BCDF KGPPCDF GHJIII GHJ... LL.MM.
Output
Papan Awal: .AAB..F .G.BCDF KGPPCDF .GHJIII .GHJ... .LL.MM. Exit is at: 2,0 Tidak ditemukan solusi.

Test Case 3 blocking

Input
6 6 1 ..PP..K .B.... .B.... ....
Output
Solution found! Number of moves: 1 Nodes visited: 6 Execution time: 4 ms Papan Awal ..PP..K .B..... .B..... .....

```
.......
.......

Gerakan 1: P-right-to-exit
.....PK
.B.....
.B.....
.......
.......
.......
```

Test Case 4 blocking

Input
6 6 7 AABCCD ..B..D ..BPPEK .....E ..FF.. ..GG..
Output
Solution found! Number of moves: 2 Nodes visited: 7 Execution time: 6 ms Papan Awal AABCCD. ..B..D. ..BPPEK .....E. ..FF... ..GG... Gerakan 1: E-down AABCCD. ..B..D. ..BPP.K

```
.....E.
..FF.E.
..GG...

Gerakan 2: P-right-to-exit
AABCCD.
..B..D.
..B..PK
.....E.
..FF.E.
..GG...
```

Test Case 5 blocking

Input
4 5 6   K FAAAA F.BBP F.EEP SS.GG
Output
Solution found! Number of moves: 3 Nodes visited: 16 Execution time: 8 ms Papan Awal ....K FAAAA F.BBP F.EEP SS.GG ..... Gerakan 1: S-right ....K FAAAA

F.BBP
F.EEP
.SSGG
.....

Gerakan 2: F-down
....K
.AAAA
F.BBP
F.EEP
FSSGG
.....

Gerakan 3: A-left
....K
AAAA.
F.BBP
F.EEP
FSSGG
.....

Test Case 6 blocking

Input
6 6 9 .AABCC .DDBE. .FPPE.K .F.... ..GHHH IIG...
Output
Solution found! Number of moves: 3 Nodes visited: 11 Execution time: 6 ms Papan Awal .AABCC. .DDBE.. .FPPE.K

```
.F.....
..GHHH.
IIG....

Gerakan 1: C-right
.AAB.CC
.DDBE..
.FPPE.K
.F.....
..GHHH.
IIG....

Gerakan 2: E-up
.AABECC
.DDBE..
.FPP..K
.F.....
..GHHH.
IIG....

Gerakan 3: P-right-to-exit
.AABECC
.DDBE..
.F...PK
.F.....
..GHHH.
IIG....
```

Test Case 7 blocking

Input
10 10 15 AAB....F.. ..B.CCCF.M D.B..T..OM D..EET..OM D.PP.T..O.K D..LLL...H .....GGGH J...NNN..H JIII..... J.....

Output

```
Solution found!
Number of moves: 8
Nodes visited: 41368
Execution time: 7518 ms

Papan Awal
AAB....F...
..B.CCCF.M.
D.B..T..OM.
D..EET..OM.
D.PP.T..O.K
D..LLL...H.
......GGGH.
J...NNN..H.
JIII.......
J.........

Gerakan 1: O-up
AAB....F...
..B.CCCFOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D..LLL...H.
......GGGH.
J...NNN..H.
JIII.......
J.........

Gerakan 2: N-right
AAB....F...
..B.CCCFOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D..LLL...H.
......GGGH.
J....NNN.H.
JIII.......
J.........

Gerakan 3: N-right
AAB....F...
..B.CCCFOM.
```

D.B..T..OM.
D..EET..OM.
D.PP.T....K
D..LLL...H.
.....GGGH.
J.....NNNH.
JIII.....
J.....

Gerakan 4: L-left
AAB....F...
..B.CCCFOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D.LLL....H.
.....GGGH.
J.....NNNH.
JIII.....
J.....

Gerakan 5: T-down
AAB....F...
..B.CCCFOM.
D.B.....OM.
D..EET..OM.
D.PP.T....K
D.LLLT...H.
.....GGGH.
J.....NNNH.
JIII.....
J.....

Gerakan 6: T-down
AAB....F...
..B.CCCFOM.
D.B.....OM.
D..EE...OM.
D.PP.T....K
D.LLLT...H.
.....TGGGH.
J.....NNNH.
JIII.....
J.....

Gerakan 7: T-down

AAB....F...
..B.CCCFOM.
D.B.....OM.
D..EE...OM.
D.PP.....K
D.LLLT...H.
.....TGGGH.
J....TNNNH.
JIII.......
J.........

Gerakan 8: P-right-to-exit
AAB....F...
..B.CCCFOM.
D.B.....OM.
D..EE...OM.
D........PK
D.LLLT...H.
.....TGGGH.
J....TNNNH.
JIII.......
J.........

Test Case 8 blocking

Input
.7 7 9 KKK.... RRRRQ.. K..NQMPP W.NLLM. W.X..M. ..X.... ..OOO..
Output
pintu keluar K tidak valid, ditemukan: 3 Terjadi kesalahan saat menjalankan program:

Test Case 9 blocking

Input
5 5 6   K A.ABB A.CCC DDDF. ...FP .EEEP
Output
Terjadi kesalahan saat menjalankan program:

Test Case 1 distance

Input
6 7 12 AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.
Output
Solution found! Number of moves: 9 Nodes visited: 105 Execution time: 16 ms Papan Awal AAB..F. ..BCDF. GPPCDFK GH.III. GHJ.... LLJMM.. Gerakan 1: C-up AABC.F. ..BCDF. GPP.DFK

GH.III.
GHJ....
LLJMM..

Gerakan 2: D-up
AABCDF.
..BCDF.
GPP..FK
GH.III.
GHJ....
LLJMM..

Gerakan 3: P-right
AABCDF.
..BCDF.
G.PP.FK
GH.III.
GHJ....
LLJMM..

Gerakan 4: P-right
AABCDF.
..BCDF.
G..PPFK
GH.III.
GHJ....
LLJMM..

Gerakan 5: I-left
AABCDF.
..BCDF.
G..PPFK
GHIII..
GHJ....
LLJMM..

Gerakan 6: F-down
AABCD..
..BCDF.
G..PPFK
GHIIIF.
GHJ....
LLJMM..

Gerakan 7: F-down
AABCD..

..BCD..
G..PPFK
GHIIIF.
GHJ..F.
LLJMM..

Gerakan 8: F-down
AABCD..
..BCD..
G..PP.K
GHIIIF.
GHJ..F.
LLJMMF.

Gerakan 9: P-right-to-exit
AABCD..
..BCD..
G....PK
GHIIIF.
GHJ..F.
LLJMMF.

Test Case 2 distance

Input
6 6 11 AAB..F G.BCDF KGPPCDF GHJIII GHJ... LL.MM.
Output
Papan Awal: .AAB..F .G.BCDF KGPPCDF .GHJIII .GHJ... .LL.MM.

Exit is at: 2,0 Tidak ditemukan solusi.

Test Case 3 distance

Input
6 6 1 ..PP..K .B.... .B.... ....
Output
Solution found! Number of moves: 1 Nodes visited: 2 Execution time: 3 ms Papan Awal ..PP..K .B..... .B..... .... Gerakan 1: P-right-to-exit .....PK .B..... .B..... ....

Test Case 4 distance

Input
6 6 7

AABCCD ..B..D ..BPPEK .....E ..FF.. ..GG..
Output
Solution found! Number of moves: 2 Nodes visited: 3 Execution time: 5 ms Papan Awal AABCCD. ..B..D. ..BPPEK .....E. ..FF... ..GG... Gerakan 1: E-down AABCCD. ..B..D. ..BPP.K .....E. ..FF.E. ..GG... Gerakan 2: P-right-to-exit AABCCD. ..B..D. ..B..PK .....E. ..FF.E. ..GG...

Test Case 5 distance

Input
4 5 6

```
   K
FAAAA
F.BBP
F.EEP
SS.GG
```

Output

```
Solution found!
Number of moves: 3
Nodes visited: 15
Execution time: 6 ms

Papan Awal
....K
FAAAA
F.BBP
F.EEP
SS.GG
.....

Gerakan 1: S-right
....K
FAAAA
F.BBP
F.EEP
.SSGG
.....

Gerakan 2: F-down
....K
.AAAA
F.BBP
F.EEP
FSSGG
.....

Gerakan 3: A-left
....K
AAAA.
F.BBP
F.EEP
FSSGG
```

.....

Test Case 6 distance

Input
6 6 9 .AABCC .DDBE. .FPPE.K .F.... ..GHHH IIG...
Output
Solution found! Number of moves: 3 Nodes visited: 6 Execution time: 5 ms Papan Awal .AABCC. .DDBE.. .FPPE.K .F..... ..GHHH. IIG.... Gerakan 1: C-right .AAB.CC .DDBE.. .FPPE.K .F..... ..GHHH. IIG.... Gerakan 2: E-up .AABECC .DDBE.. .FPP..K .F..... ..GHHH. IIG....

Gerakan 3: P-right-to-exit
.AABECC
.DDBE..
.F...PK
.F.....
..GHHH.
IIG....

Test Case 7 distance

Input
10 10 15 AAB....F.. ..B.CCCF.M D.B..T..OM D..EET..OM D.PP.T..O.K D..LLL...H .....GGGH J...NNN..H JIII..... J.....
Output
Solution found! Number of moves: 8 Nodes visited: 2310 Execution time: 396 ms Papan Awal AAB....F... ..B.CCCF.M. D.B..T..OM. D..EET..OM. D.PP.T..O.K D..LLL...H. .....GGGH. J...NNN..H. JIII..... J..... Gerakan 1: O-up

AAB....F...
..B.CCCFOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.........

Gerakan 2: P-right
AAB....F...
..B.CCCFOM.
D.B..T..OM.
D..EET..OM.
D..PPT....K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.........

Gerakan 3: B-down
AA.....F...
..B.CCCFOM.
D.B..T..OM.
D.BEET..OM.
D..PPT....K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.........

Gerakan 4: B-down
AA.....F...
....CCCFOM.
D.B..T..OM.
D.BEET..OM.
D.BPPT....K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.........

Gerakan 5: C-left
AA.....F...
...CCC.FOM.
D.B..T..OM.
D.BEET..OM.
D.BPPT....K
D..LLL...H.
......GGGH.
J...NNN..H.
JIII.......
J.........

Gerakan 6: C-left
AA.....F...
..CCC..FOM.
D.B..T..OM.
D.BEET..OM.
D.BPPT....K
D..LLL...H.
......GGGH.
J...NNN..H.
JIII.......
J.........

Gerakan 7: T-up
AA.....F...
..CCCT.FOM.
D.B..T..OM.
D.BEET..OM.
D.BPP.....K
D..LLL...H.
......GGGH.
J...NNN..H.
JIII.......
J.........

Gerakan 8: P-right-to-exit
AA.....F...
..CCCT.FOM.
D.B..T..OM.
D.BEET..OM.
D.B......PK
D..LLL...H.
......GGGH.
J...NNN..H.

JIII.....
J.....

Test Case 8 distance

Input
7 7 9 KKK.... RRRRQ.. K..NQMPP W.NLLM. W.X..M. ..X.... ..OOO..
Output
pintu keluar K tidak valid, ditemukan: 3

Test Case 9 distance

Input
5 5 6   K A.ABB A.CCC DDDF. ...FP .EEEP
Output
Solution found! Number of moves: 2 Nodes visited: 3 Execution time: 5 ms Papan Awal ....K A..BB

A.CCC
DDDF.
...FP
.EEEP

Gerakan 1: B-left
....K
A.BB.
A.CCC
DDDF.
...FP
.EEEP

Gerakan 2: C-left
....K
A.BB.
ACCC.
DDDF.
...FP
.EEEP

3.3.1. GreedyByFirst Algorithm

Test Case 1 blocking

Input
6 7 12 AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.
Output
Solution found! Number of moves: 47 Nodes visited: 1381 Execution time: 87 ms Papan Awal AAB..F.

..BCDF.
GPPCDFK
GH.III.
GHJ....
LLJMM..

Gerakan 1: C-up
AABC.F.
..BCDF.
GPP.DFK
GH.III.
GHJ....
LLJMM..

Gerakan 2: D-up
AABCDF.
..BCDF.
GPP..FK
GH.III.
GHJ....
LLJMM..

Gerakan 3: M-right
AABCDF.
..BCDF.
GPP..FK
GH.III.
GHJ....
LLJ.MM.

Gerakan 4: J-up
AABCDF.
..BCDF.
GPP..FK
GHJIII.
GHJ....
LL..MM.

Gerakan 5: M-right
AABCDF.
..BCDF.
GPP..FK
GHJIII.
GHJ....
LL...MM

Gerakan 6: P-right
AABCDF.
..BCDF.
G.PP.FK
GHJIII.
GHJ....
LL...MM

Gerakan 7: H-up
AABCDF.
..BCDF.
GHPP.FK
GHJIII.
G.J....
LL...MM

Gerakan 8: G-up
AABCDF.
G.BCDF.
GHPP.FK
GHJIII.
..J....
LL...MM

Gerakan 9: L-right
AABCDF.
G.BCDF.
GHPP.FK
GHJIII.
..J....
.LL..MM

Gerakan 10: M-left
AABCDF.
G.BCDF.
GHPP.FK
GHJIII.
..J....
.LL.MM.

Gerakan 11: M-left
AABCDF.
G.BCDF.
GHPP.FK
GHJIII.
..J....

.LLMM..

Gerakan 12: L-left
AABCDF.
G.BCDF.
GHPP.FK
GHJIII.
..J....
LL.MM..

Gerakan 13: M-left
AABCDF.
G.BCDF.
GHPP.FK
GHJIII.
..J....
LLMM...

Gerakan 14: I-right
AABCDF.
G.BCDF.
GHPP.FK
GHJ.III
..J....
LLMM...

Gerakan 15: H-down
AABCDF.
G.BCDF.
G.PP.FK
GHJ.III
.HJ....
LLMM...

Gerakan 16: P-right
AABCDF.
G.BCDF.
G..PPFK
GHJ.III
.HJ....
LLMM...

Gerakan 17: M-right
AABCDF.
G.BCDF.
G..PPFK

GHJ.III
.HJ....
LL.MM..

Gerakan 18: H-up
AABCDF.
G.BCDF.
GH.PPFK
GHJ.III
..J....
LL.MM..

Gerakan 19: H-up
AABCDF.
GHBCDF.
GH.PPFK
G.J.III
..J....
LL.MM..

Gerakan 20: G-down
AABCDF.
.HBCDF.
GH.PPFK
G.J.III
G.J....
LL.MM..

Gerakan 21: J-down
AABCDF.
.HBCDF.
GH.PPFK
G...III
G.J....
LLJMM..

Gerakan 22: M-right
AABCDF.
.HBCDF.
GH.PPFK
G...III
G.J....
LLJ.MM.

Gerakan 23: P-left
AABCDF.

.HBCDF.
GHPP.FK
G...III
G.J....
LLJ.MM.

Gerakan 24: I-left
AABCDF.
.HBCDF.
GHPP.FK
G..III.
G.J....
LLJ.MM.

Gerakan 25: M-right
AABCDF.
.HBCDF.
GHPP.FK
G..III.
G.J....
LLJ..MM

Gerakan 26: I-left
AABCDF.
.HBCDF.
GHPP.FK
G.III..
G.J....
LLJ..MM

Gerakan 27: M-left
AABCDF.
.HBCDF.
GHPP.FK
G.III..
G.J....
LLJ.MM.

Gerakan 28: M-left
AABCDF.
.HBCDF.
GHPP.FK
G.III..
G.J....
LLJMM..

Gerakan 29: I-left
AABCDF.
.HBCDF.
GHPP.FK
GIII...
G.J....
LLJMM..

Gerakan 30: M-right
AABCDF.
.HBCDF.
GHPP.FK
GIII...
G.J....
LLJ.MM.

Gerakan 31: G-up
AABCDF.
GHBCDF.
GHPP.FK
GIII...
..J....
LLJ.MM.

Gerakan 32: M-right
AABCDF.
GHBCDF.
GHPP.FK
GIII...
..J....
LLJ..MM

Gerakan 33: F-down
AABCD..
GHBCDF.
GHPP.FK
GIII.F.
..J....
LLJ..MM

Gerakan 34: M-left
AABCD..
GHBCDF.
GHPP.FK
GIII.F.
..J....

LLJ.MM.

Gerakan 35: I-right
AABCD..
GHBCDF.
GHPP.FK
G.IIIF.
..J....
LLJ.MM.

Gerakan 36: M-right
AABCD..
GHBCDF.
GHPP.FK
G.IIIF.
..J....
LLJ..MM

Gerakan 37: H-down
AABCD..
G.BCDF.
GHPP.FK
GHIIIF.
..J....
LLJ..MM

Gerakan 38: M-left
AABCD..
G.BCDF.
GHPP.FK
GHIIIF.
..J....
LLJ.MM.

Gerakan 39: H-down
AABCD..
G.BCDF.
G.PP.FK
GHIIIF.
.HJ....
LLJ.MM.

Gerakan 40: M-right
AABCD..
G.BCDF.
G.PP.FK

GHIIIF.
.HJ....
LLJ..MM

Gerakan 41: G-down
AABCD..
..BCDF.
G.PP.FK
GHIIIF.
GHJ....
LLJ..MM

Gerakan 42: M-left
AABCD..
..BCDF.
G.PP.FK
GHIIIF.
GHJ....
LLJ.MM.

Gerakan 43: M-left
AABCD..
..BCDF.
G.PP.FK
GHIIIF.
GHJ....
LLJMM..

Gerakan 44: G-up
AABCD..
G.BCDF.
G.PP.FK
GHIIIF.
.HJ....
LLJMM..

Gerakan 45: F-down
AABCD..
G.BCD..
G.PP.FK
GHIIIF.
.HJ..F.
LLJMM..

Gerakan 46: F-down
AABCD..

G.BCD..
G.PP..K
GHIIIF.
.HJ..F.
LLJMMF.

Gerakan 47: P-right-to-exit
AABCD..
G.BCD..
G....PK
GHIIIF.
.HJ..F.
LLJMMF.

Test Case 2 blocking

Input
6 6 11 AAB..F G.BCDF KGPPCDF GHJIII GHJ... LL.MM.
Output
Papan Awal: .AAB..F .G.BCDF KGPPCDF .GHJIII .GHJ... .LL.MM. Exit is at: 2,0 Tidak ditemukan solusi.

Test Case 3 blocking

Input
6 6 1

..PP..K
.B....
.B....
.....
.....
.....

Output

Solution found!
Number of moves: 4
Nodes visited: 9
Execution time: 5 ms

Papan Awal
..PP..K
.B.....
.B.....
....
....
....

Gerakan 1: B-down
..PP..K
....
.B.....
.B.....
....
....

Gerakan 2: B-down
..PP..K
....
....
.B.....
.B.....
....

Gerakan 3: B-down
..PP..K
....
....
....
.B.....
.B.....

Gerakan 4: P-right-to-exit
.....PK
.....
.....
.....
.B.....
.B.....

Test Case 4 blocking

Input
6 6 7 AABCCD ..B..D ..BPPEK .....E ..FF.. ..GG..
Output
Solution found! Number of moves: 4 Nodes visited: 13 Execution time: 11 ms Papan Awal AABCCD. ..B..D. ..BPPEK .....E. ..FF... ..GG... Gerakan 1: E-down AABCCD. ..B..D. ..BPP.K .....E. ..FF.E. ..GG... Gerakan 2: P-right

AABCCD.
..B..D.
..B.PPK
.....E.
..FF.E.
..GG...

Gerakan 3: G-right
AABCCD.
..B..D.
..B.PPK
.....E.
..FF.E.
...GG..

Gerakan 4: P-right
AABCCD.
..B..D.
..B..PK
.....E.
..FF.E.
...GG..

Test Case 5 blocking

Input
4 5 6 K FAAAA F.BBP F.EEP SS.GG
Output
Solution found! Number of moves: 10 Nodes visited: 20 Execution time: 7 ms Papan Awal

....K
FAAAA
F.BBP
F.EEP
SS.GG
.....

Gerakan 1: G-left
....K
FAAAA
F.BBP
F.EEP
SSGG.
.....

Gerakan 2: F-up
F...K
FAAAA
F.BBP
..EEP
SSGG.
.....

Gerakan 3: G-right
F...K
FAAAA
F.BBP
..EEP
SS.GG
.....

Gerakan 4: E-left
F...K
FAAAA
F.BBP
.EE.P
SS.GG
.....

Gerakan 5: G-left
F...K
FAAAA
F.BBP
.EE.P
SSGG.
.....

Gerakan 6: F-down
....K
FAAAA
F.BBP
FEE.P
SSGG.
.....

Gerakan 7: G-right
....K
FAAAA
F.BBP
FEE.P
SS.GG
.....

Gerakan 8: S-right
....K
FAAAA
F.BBP
FEE.P
.SSGG
.....

Gerakan 9: F-down
....K
.AAAA
F.BBP
FEE.P
FSSGG
.....

Gerakan 10: A-left
....K
AAAA.
F.BBP
FEE.P
FSSGG
.....

Test Case 6 blocking

Input

```
6 6
9
.AABCC
.DDBE.
.FPPE.K
.F....
..GHHH
IIG...
```

Output

```
Solution found!
Number of moves: 114
Nodes visited: 228
Execution time: 29 ms

Papan Awal
.AABCC.
.DDBE..
.FPPE.K
.F.....
..GHHH.
IIG....

Gerakan 1: H-right
.AABCC.
.DDBE..
.FPPE.K
.F.....
..G.HHH
IIG....

Gerakan 2: G-up
.AABCC.
.DDBE..
.FPPE.K
.FG....
..G.HHH
II.....

Gerakan 3: I-right
.AABCC.
.DDBE..
.FPPE.K
.FG....
```

..G.HHH
.II....

Gerakan 4: I-right
.AABCC.
.DDBE..
.FPPE.K
.FG....
..G.HHH
..II...

Gerakan 5: I-right
.AABCC.
.DDBE..
.FPPE.K
.FG....
..G.HHH
...II..

Gerakan 6: I-right
.AABCC.
.DDBE..
.FPPE.K
.FG....
..G.HHH
....II.

Gerakan 7: I-right
.AABCC.
.DDBE..
.FPPE.K
.FG....
..G.HHH
.....II

Gerakan 8: H-left
.AABCC.
.DDBE..
.FPPE.K
.FG....
..GHHH.
.....II

Gerakan 9: I-left
.AABCC.
.DDBE..

.FPPE.K
.FG....
..GHHH.
....II.

Gerakan 10: I-left
.AABCC.
.DDBE..
.FPPE.K
.FG....
..GHHH.
...II..

Gerakan 11: I-left
.AABCC.
.DDBE..
.FPPE.K
.FG....
..GHHH.
..II...

Gerakan 12: I-left
.AABCC.
.DDBE..
.FPPE.K
.FG....
..GHHH.
.II....

Gerakan 13: I-left
.AABCC.
.DDBE..
.FPPE.K
.FG....
..GHHH.
II.....

Gerakan 14: F-down
.AABCC.
.DDBE..
..PPE.K
.FG....
.FGHHH.
II.....

Gerakan 15: I-right

.AABCC.
.DDBE..
..PPE.K
.FG....
.FGHHH.
.II....

Gerakan 16: I-right
.AABCC.
.DDBE..
..PPE.K
.FG....
.FGHHH.
..II...

Gerakan 17: I-right
.AABCC.
.DDBE..
..PPE.K
.FG....
.FGHHH.
...II..

Gerakan 18: I-right
.AABCC.
.DDBE..
..PPE.K
.FG....
.FGHHH.
....II.

Gerakan 19: I-right
.AABCC.
.DDBE..
..PPE.K
.FG....
.FGHHH.
.....II

Gerakan 20: H-right
.AABCC.
.DDBE..
..PPE.K
.FG....
.FG.HHH
.....II

Gerakan 21: I-left
.AABCC.
.DDBE..
..PPE.K
.FG....
.FG.HHH
....II.

Gerakan 22: I-left
.AABCC.
.DDBE..
..PPE.K
.FG....
.FG.HHH
...II..

Gerakan 23: I-left
.AABCC.
.DDBE..
..PPE.K
.FG....
.FG.HHH
..II...

Gerakan 24: I-left
.AABCC.
.DDBE..
..PPE.K
.FG....
.FG.HHH
.II....

Gerakan 25: I-left
.AABCC.
.DDBE..
..PPE.K
.FG....
.FG.HHH
II.....

Gerakan 26: G-down
.AABCC.
.DDBE..
..PPE.K
.F.....

.FG.HHH
IIG....

Gerakan 27: H-left
.AABCC.
.DDBE..
..PPE.K
.F.....
.FGHHH.
IIG....

Gerakan 28: E-down
.AABCC.
.DDB...
..PPE.K
.F..E..
.FGHHH.
IIG....

Gerakan 29: H-right
.AABCC.
.DDB...
..PPE.K
.F..E..
.FG.HHH
IIG....

Gerakan 30: G-up
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FG.HHH
II.....

Gerakan 31: I-right
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FG.HHH
.II....

Gerakan 32: I-right
.AABCC.
.DDB...

..PPE.K
.FG.E..
.FG.HHH
..II...

Gerakan 33: I-right
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FG.HHH
...II..

Gerakan 34: I-right
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FG.HHH
....II.

Gerakan 35: I-right
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FG.HHH
.....II

Gerakan 36: H-left
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FGHHH.
.....II

Gerakan 37: I-left
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FGHHH.
....II.

Gerakan 38: I-left

.AABCC.
.DDB...
..PPE.K
.FG.E..
.FGHHH.
...II..

Gerakan 39: I-left
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FGHHH.
..II...

Gerakan 40: I-left
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FGHHH.
.II....

Gerakan 41: I-left
.AABCC.
.DDB...
..PPE.K
.FG.E..
.FGHHH.
II.....

Gerakan 42: F-up
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..GHHH.
II.....

Gerakan 43: I-right
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..GHHH.
.II....

Gerakan 44: I-right
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..GHHH.
..II...

Gerakan 45: I-right
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..GHHH.
...II..

Gerakan 46: I-right
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..GHHH.
....II.

Gerakan 47: I-right
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..GHHH.
.....II

Gerakan 48: H-right
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..G.HHH
.....II

Gerakan 49: I-left
.AABCC.
.DDB...
.FPPE.K
.FG.E..

..G.HHH
....II.

Gerakan 50: I-left
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..G.HHH
...II..

Gerakan 51: I-left
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..G.HHH
..II...

Gerakan 52: I-left
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..G.HHH
.II....

Gerakan 53: I-left
.AABCC.
.DDB...
.FPPE.K
.FG.E..
..G.HHH
II.....

Gerakan 54: G-down
.AABCC.
.DDB...
.FPPE.K
.F..E..
..G.HHH
IIG....

Gerakan 55: H-left
.AABCC.
.DDB...

.FPPE.K
.F..E..
..GHHH.
IIG....

Gerakan 56: D-left
.AABCC.
DD.B...
.FPPE.K
.F..E..
..GHHH.
IIG....

Gerakan 57: H-right
.AABCC.
DD.B...
.FPPE.K
.F..E..
..G.HHH
IIG....

Gerakan 58: G-up
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..G.HHH
II.....

Gerakan 59: I-right
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..G.HHH
.II....

Gerakan 60: I-right
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..G.HHH
..II...

Gerakan 61: I-right

.AABCC.
DD.B...
.FPPE.K
.FG.E..
..G.HHH
...II..

Gerakan 62: I-right
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..G.HHH
....II.

Gerakan 63: I-right
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..G.HHH
.....II

Gerakan 64: H-left
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..GHHH.
.....II

Gerakan 65: I-left
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..GHHH.
....II.

Gerakan 66: I-left
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..GHHH.
...II..

Gerakan 67: I-left
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..GHHH.
..II...

Gerakan 68: I-left
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..GHHH.
.II....

Gerakan 69: I-left
.AABCC.
DD.B...
.FPPE.K
.FG.E..
..GHHH.
II.....

Gerakan 70: F-down
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FGHHH.
II.....

Gerakan 71: I-right
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FGHHH.
.II....

Gerakan 72: I-right
.AABCC.
DD.B...
..PPE.K
.FG.E..

.FGHHH.
..II...

Gerakan 73: I-right
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FGHHH.
...II..

Gerakan 74: I-right
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FGHHH.
....II.

Gerakan 75: I-right
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FGHHH.
.....II

Gerakan 76: H-right
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FG.HHH
.....II

Gerakan 77: I-left
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FG.HHH
....II.

Gerakan 78: I-left
.AABCC.
DD.B...

..PPE.K
.FG.E..
.FG.HHH
...II..

Gerakan 79: I-left
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FG.HHH
..II...

Gerakan 80: I-left
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FG.HHH
.II....

Gerakan 81: I-left
.AABCC.
DD.B...
..PPE.K
.FG.E..
.FG.HHH
II.....

Gerakan 82: G-down
.AABCC.
DD.B...
..PPE.K
.F..E..
.FG.HHH
IIG....

Gerakan 83: H-left
.AABCC.
DD.B...
..PPE.K
.F..E..
.FGHHH.
IIG....

Gerakan 84: E-up

.AABCC.
DD.BE..
..PPE.K
.F.....
.FGHHH.
IIG....

Gerakan 85: H-right
.AABCC.
DD.BE..
..PPE.K
.F.....
.FG.HHH
IIG....

Gerakan 86: G-up
.AABCC.
DD.BE..
..PPE.K
.FG....
.FG.HHH
II.....

Gerakan 87: I-right
.AABCC.
DD.BE..
..PPE.K
.FG....
.FG.HHH
.II....

Gerakan 88: I-right
.AABCC.
DD.BE..
..PPE.K
.FG....
.FG.HHH
..II...

Gerakan 89: I-right
.AABCC.
DD.BE..
..PPE.K
.FG....
.FG.HHH
...II..

Gerakan 90: I-right
.AABCC.
DD.BE..
..PPE.K
.FG....
.FG.HHH
....II.

Gerakan 91: I-right
.AABCC.
DD.BE..
..PPE.K
.FG....
.FG.HHH
.....II

Gerakan 92: H-left
.AABCC.
DD.BE..
..PPE.K
.FG....
.FGHHH.
.....II

Gerakan 93: I-left
.AABCC.
DD.BE..
..PPE.K
.FG....
.FGHHH.
....II.

Gerakan 94: I-left
.AABCC.
DD.BE..
..PPE.K
.FG....
.FGHHH.
...II..

Gerakan 95: I-left
.AABCC.
DD.BE..
..PPE.K
.FG....

.FGHHH.
..II...

Gerakan 96: I-left
.AABCC.
DD.BE..
..PPE.K
.FG....
.FGHHH.
.II....

Gerakan 97: I-left
.AABCC.
DD.BE..
..PPE.K
.FG....
.FGHHH.
II.....

Gerakan 98: F-up
.AABCC.
DD.BE..
.FPPE.K
.FG....
..GHHH.
II.....

Gerakan 99: I-right
.AABCC.
DD.BE..
.FPPE.K
.FG....
..GHHH.
.II....

Gerakan 100: I-right
.AABCC.
DD.BE..
.FPPE.K
.FG....
..GHHH.
..II...

Gerakan 101: I-right
.AABCC.
DD.BE..

.FPPE.K
.FG....
..GHHH.
...II..

Gerakan 102: I-right
.AABCC.
DD.BE..
.FPPE.K
.FG....
..GHHH.
....II.

Gerakan 103: I-right
.AABCC.
DD.BE..
.FPPE.K
.FG....
..GHHH.
.....II

Gerakan 104: H-right
.AABCC.
DD.BE..
.FPPE.K
.FG....
..G.HHH
.....II

Gerakan 105: I-left
.AABCC.
DD.BE..
.FPPE.K
.FG....
..G.HHH
....II.

Gerakan 106: I-left
.AABCC.
DD.BE..
.FPPE.K
.FG....
..G.HHH
...II..

Gerakan 107: I-left

.AABCC.
DD.BE..
.FPPE.K
.FG....
..G.HHH
..II...

Gerakan 108: I-left
.AABCC.
DD.BE..
.FPPE.K
.FG....
..G.HHH
.II....

Gerakan 109: I-left
.AABCC.
DD.BE..
.FPPE.K
.FG....
..G.HHH
II.....

Gerakan 110: G-down
.AABCC.
DD.BE..
.FPPE.K
.F.....
..G.HHH
IIG....

Gerakan 111: H-left
.AABCC.
DD.BE..
.FPPE.K
.F.....
..GHHH.
IIG....

Gerakan 112: C-right
.AAB.CC
DD.BE..
.FPPE.K
.F.....
..GHHH.
IIG....

Gerakan 113: E-up
.AABECC
DD.BE..
.FPP..K
.F.....
..GHHH.
IIG....

Gerakan 114: P-right-to-exit
.AABECC
DD.BE..
.F...PK
.F.....
..GHHH.
IIG....

Test Case 7 blocking

Input
10 10 15 AAB....F.. ..B.CCCF.M D.B..T..OM D..EET..OM D.PP.T..O.K D..LLL...H .....GGGH J...NNN..H JIII..... J.....
Output
Solution found! Number of moves: 29 Nodes visited: 2089 Execution time: 305 ms Papan Awal AAB....F... ..B.CCCF.M.

D.B..T..OM.
D..EET..OM.
D.PP.T..O.K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.....

Gerakan 1: O-up
AAB....F...
..B.CCCFOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.....

Gerakan 2: C-left
AAB....F...
..BCCC.FOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.....

Gerakan 3: G-left
AAB....F...
..BCCC.FOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D..LLL...H.
.....GGG.H.
J...NNN..H.
JIII.....
J.....

Gerakan 4: P-right

AAB....F...
..BCCC.FOM.
D.B..T..OM.
D..EET..OM.
D..PPT....K
D..LLL...H.
.....GGG.H.
J...NNN..H.
JIII.......
J.........

Gerakan 5: I-right
AAB....F...
..BCCC.FOM.
D.B..T..OM.
D..EET..OM.
D..PPT....K
D..LLL...H.
.....GGG.H.
J...NNN..H.
J.III......
J.........

Gerakan 6: P-left
AAB....F...
..BCCC.FOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D..LLL...H.
.....GGG.H.
J...NNN..H.
J.III......
J.........

Gerakan 7: L-left
AAB....F...
..BCCC.FOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D.LLL....H.
.....GGG.H.
J...NNN..H.
J.III......
J.........

Gerakan 8: I-right
AAB....F...
..BCCC.FOM.
D.B..T..OM.
D..EET..OM.
D.PP.T....K
D.LLL....H.
.....GGG.H.
J...NNN..H.
J..III.....
J.........

Gerakan 9: T-down
AAB....F...
..BCCC.FOM.
D.B.....OM.
D..EET..OM.
D.PP.T....K
D.LLLT...H.
.....GGG.H.
J...NNN..H.
J..III.....
J.........

Gerakan 10: P-right
AAB....F...
..BCCC.FOM.
D.B.....OM.
D..EET..OM.
D..PPT....K
D.LLLT...H.
.....GGG.H.
J...NNN..H.
J..III.....
J.........

Gerakan 11: T-up
AAB....F...
..BCCC.FOM.
D.B..T..OM.
D..EET..OM.
D..PPT....K
D.LLL....H.
.....GGG.H.
J...NNN..H.

J..III.....
J.....

Gerakan 12: O-up
AAB....FO..
..BCCC.FOM.
D.B..T..OM.
D..EET...M.
D..PPT....K
D.LLL....H.
.....GGG.H.
J...NNN..H.
J..III.....
J.....

Gerakan 13: T-down
AAB....FO..
..BCCC.FOM.
D.B.....OM.
D..EET...M.
D..PPT....K
D.LLLT...H.
.....GGG.H.
J...NNN..H.
J..III.....
J.....

Gerakan 14: P-left
AAB....FO..
..BCCC.FOM.
D.B.....OM.
D..EET...M.
D.PP.T....K
D.LLLT...H.
.....GGG.H.
J...NNN..H.
J..III.....
J.....

Gerakan 15: T-up
AAB....FO..
..BCCC.FOM.
D.B..T..OM.
D..EET...M.
D.PP.T....K
D.LLL....H.

.....GGG.H.
J...NNN..H.
J..III.....
J.....

Gerakan 16: P-left
AAB....FO..
..BCCC.FOM.
D.B..T..OM.
D..EET...M.
DPP..T....K
D.LLL....H.
.....GGG.H.
J...NNN..H.
J..III.....
J.....

Gerakan 17: T-down
AAB....FO..
..BCCC.FOM.
D.B.....OM.
D..EET...M.
DPP..T....K
D.LLLT...H.
.....GGG.H.
J...NNN..H.
J..III.....
J.....

Gerakan 18: N-right
AAB....FO..
..BCCC.FOM.
D.B.....OM.
D..EET...M.
DPP..T....K
D.LLLT...H.
.....GGG.H.
J....NNN.H.
J..III.....
J.....

Gerakan 19: M-up
AAB....FOM.
..BCCC.FOM.
D.B.....OM.
D..EET.....

DPP..T....K
D.LLLT...H.
.....GGG.H.
J....NNN.H.
J..III.....
J.........

Gerakan 20: O-down
AAB....F.M.
..BCCC.FOM.
D.B.....OM.
D..EET..O..
DPP..T....K
D.LLLT...H.
.....GGG.H.
J....NNN.H.
J..III.....
J.........

Gerakan 21: T-up
AAB....F.M.
..BCCC.FOM.
D.B..T..OM.
D..EET..O..
DPP..T....K
D.LLL....H.
.....GGG.H.
J....NNN.H.
J..III.....
J.........

Gerakan 22: P-right
AAB....F.M.
..BCCC.FOM.
D.B..T..OM.
D..EET..O..
D.PP.T....K
D.LLL....H.
.....GGG.H.
J....NNN.H.
J..III.....
J.........

Gerakan 23: H-down
AAB....F.M.
..BCCC.FOM.

D.B..T..OM.
D..EET..O..
D.PP.T....K
D.LLL.....
.....GGG.H.
J....NNN.H.
J..III...H.
J.........

Gerakan 24: G-right
AAB....F.M.
..BCCC.FOM.
D.B..T..OM.
D..EET..O..
D.PP.T....K
D.LLL.....
.....GGGH.
J....NNN.H.
J..III...H.
J.........

Gerakan 25: N-right
AAB....F.M.
..BCCC.FOM.
D.B..T..OM.
D..EET..O..
D.PP.T....K
D.LLL.....
.....GGGH.
J.....NNNH.
J..III...H.
J.........

Gerakan 26: T-down
AAB....F.M.
..BCCC.FOM.
D.B.....OM.
D..EET..O..
D.PP.T....K
D.LLLT.....
.....GGGH.
J.....NNNH.
J..III...H.
J.........

Gerakan 27: T-down

AAB....F.M.
..BCCC.FOM.
D.B.....OM.
D..EE...O..
D.PP.T....K
D.LLLT.....
.....TGGGH.
J.....NNNH.
J..III...H.
J.........

Gerakan 28: T-down
AAB....F.M.
..BCCC.FOM.
D.B.....OM.
D..EE...O..
D.PP......K
D.LLLT.....
.....TGGGH.
J....TNNNH.
J..III...H.
J.........

Gerakan 29: P-right-to-exit
AAB....F.M.
..BCCC.FOM.
D.B.....OM.
D..EE...O..
D........PK
D.LLLT.....
.....TGGGH.
J....TNNNH.
J..III...H.
J.........

Test Case 8 blocking

Input
.7 7 9 KKK.... RRRRQ.. K..NQMPP W.NLLM.

W.X..M. ..X.... ..OOO..
Output
pintu keluar K tidak valid, ditemukan: 3

Test Case 9 blocking

Input
5 5 6   K A.ABB A.CCC DDDF. ...FP .EEEP
Output
Solution found! Number of moves: 2 Nodes visited: 3 Execution time: 5 ms Papan Awal ....K A..BB A.CCC DDDF. ...FP .EEEP Gerakan 1: B-left ....K A.BB. A.CCC DDDF. ...FP .EEEP Gerakan 2: C-left ....K A.BB. ACCC.

DDDF.
...FP
.EEEP

Test Case 1 distance

Input
6 7 12 AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.
Output
Solution found! Number of moves: 120 Nodes visited: 1168 Execution time: 73 ms Papan Awal AAB..F. ..BCDF. GPPCDFK GH.III. GHJ.... LLJMM.. Gerakan 1: C-up AABC.F. ..BCDF. GPP.DFK GH.III. GHJ.... LLJMM.. Gerakan 2: D-up AABCDF. ..BCDF. GPP..FK GH.III.

GHJ....
LLJMM..

Gerakan 3: P-right
AABCDF.
..BCDF.
G.PP.FK
GH.III.
GHJ....
LLJMM..

Gerakan 4: P-right
AABCDF.
..BCDF.
G..PPFK
GH.III.
GHJ....
LLJMM..

Gerakan 5: B-down
AA.CDF.
..BCDF.
G.BPPFK
GH.III.
GHJ....
LLJMM..

Gerakan 6: A-right
.AACDF.
..BCDF.
G.BPPFK
GH.III.
GHJ....
LLJMM..

Gerakan 7: M-right
.AACDF.
..BCDF.
G.BPPFK
GH.III.
GHJ....
LLJ.MM.

Gerakan 8: M-right
.AACDF.
..BCDF.

G.BPPFK
GH.III.
GHJ....
LLJ..MM

Gerakan 9: I-right
.AACDF.
..BCDF.
G.BPPFK
GH..III
GHJ....
LLJ..MM

Gerakan 10: J-up
.AACDF.
..BCDF.
G.BPPFK
GHJ.III
GHJ....
LL...MM

Gerakan 11: M-left
.AACDF.
..BCDF.
G.BPPFK
GHJ.III
GHJ....
LL..MM.

Gerakan 12: L-right
.AACDF.
..BCDF.
G.BPPFK
GHJ.III
GHJ....
.LL.MM.

Gerakan 13: M-right
.AACDF.
..BCDF.
G.BPPFK
GHJ.III
GHJ....
.LL..MM

Gerakan 14: L-right

.AACDF.
..BCDF.
G.BPPFK
GHJ.III
GHJ....
..LL.MM

Gerakan 15: M-left
.AACDF.
..BCDF.
G.BPPFK
GHJ.III
GHJ....
..LLMM.

Gerakan 16: H-down
.AACDF.
..BCDF.
G.BPPFK
G.J.III
GHJ....
.HLLMM.

Gerakan 17: M-right
.AACDF.
..BCDF.
G.BPPFK
G.J.III
GHJ....
.HLL.MM

Gerakan 18: L-right
.AACDF.
..BCDF.
G.BPPFK
G.J.III
GHJ....
.H.LLMM

Gerakan 19: J-down
.AACDF.
..BCDF.
G.BPPFK
G...III
GHJ....
.HJLLMM

Gerakan 20: I-left
.AACDF.
..BCDF.
G.BPPFK
G..III.
GHJ....
.HJLLMM

Gerakan 21: J-up
.AACDF.
..BCDF.
G.BPPFK
G.JIII.
GHJ....
.H.LLMM

Gerakan 22: L-left
.AACDF.
..BCDF.
G.BPPFK
G.JIII.
GHJ....
.HLL.MM

Gerakan 23: M-left
.AACDF.
..BCDF.
G.BPPFK
G.JIII.
GHJ....
.HLLMM.

Gerakan 24: G-down
.AACDF.
..BCDF.
..BPPFK
G.JIII.
GHJ....
GHLLMM.

Gerakan 25: M-right
.AACDF.
..BCDF.
..BPPFK
G.JIII.

GHJ....
GHLL.MM

Gerakan 26: L-right
.AACDF.
..BCDF.
..BPPFK
G.JIII.
GHJ....
GH.LLMM

Gerakan 27: J-down
.AACDF.
..BCDF.
..BPPFK
G..III.
GHJ....
GHJLLMM

Gerakan 28: I-right
.AACDF.
..BCDF.
..BPPFK
G...III
GHJ....
GHJLLMM

Gerakan 29: J-up
.AACDF.
..BCDF.
..BPPFK
G.J.III
GHJ....
GH.LLMM

Gerakan 30: L-left
.AACDF.
..BCDF.
..BPPFK
G.J.III
GHJ....
GHLL.MM

Gerakan 31: M-left
.AACDF.
..BCDF.

..BPPFK
G.J.III
GHJ....
GHLLMM.

Gerakan 32: A-left
AA.CDF.
..BCDF.
..BPPFK
G.J.III
GHJ....
GHLLMM.

Gerakan 33: M-right
AA.CDF.
..BCDF.
..BPPFK
G.J.III
GHJ....
GHLL.MM

Gerakan 34: L-right
AA.CDF.
..BCDF.
..BPPFK
G.J.III
GHJ....
GH.LLMM

Gerakan 35: J-down
AA.CDF.
..BCDF.
..BPPFK
G...III
GHJ....
GHJLLMM

Gerakan 36: I-left
AA.CDF.
..BCDF.
..BPPFK
G..III.
GHJ....
GHJLLMM

Gerakan 37: J-up

AA.CDF.
..BCDF.
..BPPFK
G.JIII.
GHJ....
GH.LLMM

Gerakan 38: L-left
AA.CDF.
..BCDF.
..BPPFK
G.JIII.
GHJ....
GHLL.MM

Gerakan 39: M-left
AA.CDF.
..BCDF.
..BPPFK
G.JIII.
GHJ....
GHLLMM.

Gerakan 40: H-up
AA.CDF.
..BCDF.
..BPPFK
GHJIII.
GHJ....
G.LLMM.

Gerakan 41: M-right
AA.CDF.
..BCDF.
..BPPFK
GHJIII.
GHJ....
G.LL.MM

Gerakan 42: L-right
AA.CDF.
..BCDF.
..BPPFK
GHJIII.
GHJ....
G..LLMM

Gerakan 43: J-down
AA.CDF.
..BCDF.
..BPPFK
GH.III.
GHJ....
G.JLLMM

Gerakan 44: I-right
AA.CDF.
..BCDF.
..BPPFK
GH..III
GHJ....
G.JLLMM

Gerakan 45: J-up
AA.CDF.
..BCDF.
..BPPFK
GHJ.III
GHJ....
G..LLMM

Gerakan 46: L-left
AA.CDF.
..BCDF.
..BPPFK
GHJ.III
GHJ....
G.LL.MM

Gerakan 47: M-left
AA.CDF.
..BCDF.
..BPPFK
GHJ.III
GHJ....
G.LLMM.

Gerakan 48: L-left
AA.CDF.
..BCDF.
..BPPFK
GHJ.III

GHJ....
GLL.MM.

Gerakan 49: M-right
AA.CDF.
..BCDF.
..BPPFK
GHJ.III
GHJ....
GLL..MM

Gerakan 50: I-left
AA.CDF.
..BCDF.
..BPPFK
GHJIII.
GHJ....
GLL..MM

Gerakan 51: M-left
AA.CDF.
..BCDF.
..BPPFK
GHJIII.
GHJ....
GLL.MM.

Gerakan 52: M-left
AA.CDF.
..BCDF.
..BPPFK
GHJIII.
GHJ....
GLLMM..

Gerakan 53: I-right
AA.CDF.
..BCDF.
..BPPFK
GHJ.III
GHJ....
GLLMM..

Gerakan 54: H-up
AA.CDF.
..BCDF.

.HBPPFK
GHJ.III
G.J....
GLLMM..

Gerakan 55: M-right
AA.CDF.
..BCDF.
.HBPPFK
GHJ.III
G.J....
GLL.MM.

Gerakan 56: M-right
AA.CDF.
..BCDF.
.HBPPFK
GHJ.III
G.J....
GLL..MM

Gerakan 57: L-right
AA.CDF.
..BCDF.
.HBPPFK
GHJ.III
G.J....
G.LL.MM

Gerakan 58: M-left
AA.CDF.
..BCDF.
.HBPPFK
GHJ.III
G.J....
G.LLMM.

Gerakan 59: I-left
AA.CDF.
..BCDF.
.HBPPFK
GHJIII.
G.J....
G.LLMM.

Gerakan 60: M-right

AA.CDF.
..BCDF.
.HBPPFK
GHJIII.
G.J....
G.LL.MM

Gerakan 61: L-right
AA.CDF.
..BCDF.
.HBPPFK
GHJIII.
G.J....
G..LLMM

Gerakan 62: J-down
AA.CDF.
..BCDF.
.HBPPFK
GH.III.
G.J....
G.JLLMM

Gerakan 63: I-right
AA.CDF.
..BCDF.
.HBPPFK
GH..III
G.J....
G.JLLMM

Gerakan 64: J-up
AA.CDF.
..BCDF.
.HBPPFK
GHJ.III
G.J....
G..LLMM

Gerakan 65: G-up
AA.CDF.
..BCDF.
GHBPPFK
GHJ.III
G.J....
...LLMM

Gerakan 66: L-left
AA.CDF.
..BCDF.
GHBPPFK
GHJ.III
G.J....
..LL.MM

Gerakan 67: M-left
AA.CDF.
..BCDF.
GHBPPFK
GHJ.III
G.J....
..LLMM.

Gerakan 68: L-left
AA.CDF.
..BCDF.
GHBPPFK
GHJ.III
G.J....
.LL.MM.

Gerakan 69: M-right
AA.CDF.
..BCDF.
GHBPPFK
GHJ.III
G.J....
.LL..MM

Gerakan 70: L-left
AA.CDF.
..BCDF.
GHBPPFK
GHJ.III
G.J....
LL...MM

Gerakan 71: M-left
AA.CDF.
..BCDF.
GHBPPFK
GHJ.III

G.J....
LL..MM.

Gerakan 72: M-left
AA.CDF.
..BCDF.
GHBPPFK
GHJ.III
G.J....
LL.MM..

Gerakan 73: M-left
AA.CDF.
..BCDF.
GHBPPFK
GHJ.III
G.J....
LLMM...

Gerakan 74: I-left
AA.CDF.
..BCDF.
GHBPPFK
GHJIII.
G.J....
LLMM...

Gerakan 75: M-right
AA.CDF.
..BCDF.
GHBPPFK
GHJIII.
G.J....
LL.MM..

Gerakan 76: B-up
AABCDF.
..BCDF.
GH.PPFK
GHJIII.
G.J....
LL.MM..

Gerakan 77: J-down
AABCDF.
..BCDF.

GH.PPFK
GH.III.
G.J....
LLJMM..

Gerakan 78: I-left
AABCDF.
..BCDF.
GH.PPFK
GHIII..
G.J....
LLJMM..

Gerakan 79: M-right
AABCDF.
..BCDF.
GH.PPFK
GHIII..
G.J....
LLJ.MM.

Gerakan 80: H-down
AABCDF.
..BCDF.
G..PPFK
GHIII..
GHJ....
LLJ.MM.

Gerakan 81: M-right
AABCDF.
..BCDF.
G..PPFK
GHIII..
GHJ....
LLJ..MM

Gerakan 82: G-up
AABCDF.
G.BCDF.
G..PPFK
GHIII..
.HJ....
LLJ..MM

Gerakan 83: M-left

AABCDF.
G.BCDF.
G..PPFK
GHIII..
.HJ....
LLJ.MM.

Gerakan 84: H-up
AABCDF.
G.BCDF.
GH.PPFK
GHIII..
..J....
LLJ.MM.

Gerakan 85: M-right
AABCDF.
G.BCDF.
GH.PPFK
GHIII..
..J....
LLJ..MM

Gerakan 86: H-up
AABCDF.
GHBCDF.
GH.PPFK
G.III..
..J....
LLJ..MM

Gerakan 87: M-left
AABCDF.
GHBCDF.
GH.PPFK
G.III..
..J....
LLJ.MM.

Gerakan 88: M-left
AABCDF.
GHBCDF.
GH.PPFK
G.III..
..J....
LLJMM..

Gerakan 89: I-right
AABCDF.
GHBCDF.
GH.PPFK
G..III.
..J....
LLJMM..

Gerakan 90: M-right
AABCDF.
GHBCDF.
GH.PPFK
G..III.
..J....
LLJ.MM.

Gerakan 91: M-right
AABCDF.
GHBCDF.
GH.PPFK
G..III.
..J....
LLJ..MM

Gerakan 92: I-right
AABCDF.
GHBCDF.
GH.PPFK
G...III
..J....
LLJ..MM

Gerakan 93: G-down
AABCDF.
.HBCDF.
GH.PPFK
G...III
G.J....
LLJ..MM

Gerakan 94: B-down
AA.CDF.
.HBCDF.
GHBPPFK
G...III

G.J....
LLJ..MM

Gerakan 95: M-left
AA.CDF.
.HBCDF.
GHBPPFK
G...III
G.J....
LLJ.MM.

Gerakan 96: M-left
AA.CDF.
.HBCDF.
GHBPPFK
G...III
G.J....
LLJMM..

Gerakan 97: I-left
AA.CDF.
.HBCDF.
GHBPPFK
G..III.
G.J....
LLJMM..

Gerakan 98: I-left
AA.CDF.
.HBCDF.
GHBPPFK
G.III..
G.J....
LLJMM..

Gerakan 99: M-right
AA.CDF.
.HBCDF.
GHBPPFK
G.III..
G.J....
LLJ.MM.

Gerakan 100: M-right
AA.CDF.
.HBCDF.

GHBPPFK
G.III..
G.J....
LLJ..MM

Gerakan 101: I-left
AA.CDF.
.HBCDF.
GHBPPFK
GIII...
G.J....
LLJ..MM

Gerakan 102: M-left
AA.CDF.
.HBCDF.
GHBPPFK
GIII...
G.J....
LLJ.MM.

Gerakan 103: M-left
AA.CDF.
.HBCDF.
GHBPPFK
GIII...
G.J....
LLJMM..

Gerakan 104: G-up
AA.CDF.
GHBCDF.
GHBPPFK
GIII...
..J....
LLJMM..

Gerakan 105: F-down
AA.CD..
GHBCDF.
GHBPPFK
GIII.F.
..J....
LLJMM..

Gerakan 106: M-right

AA.CD..
GHBCDF.
GHBPPFK
GIII.F.
..J....
LLJ.MM.

Gerakan 107: M-right
AA.CD..
GHBCDF.
GHBPPFK
GIII.F.
..J....
LLJ..MM

Gerakan 108: I-right
AA.CD..
GHBCDF.
GHBPPFK
G.IIIF.
..J....
LLJ..MM

Gerakan 109: M-left
AA.CD..
GHBCDF.
GHBPPFK
G.IIIF.
..J....
LLJ.MM.

Gerakan 110: M-left
AA.CD..
GHBCDF.
GHBPPFK
G.IIIF.
..J....
LLJMM..

Gerakan 111: H-down
AA.CD..
G.BCDF.
GHBPPFK
GHIIIF.
..J....
LLJMM..

Gerakan 112: H-down
AA.CD..
G.BCDF.
G.BPPFK
GHIIIF.
.HJ....
LLJMM..

Gerakan 113: M-right
AA.CD..
G.BCDF.
G.BPPFK
GHIIIF.
.HJ....
LLJ.MM.

Gerakan 114: M-right
AA.CD..
G.BCDF.
G.BPPFK
GHIIIF.
.HJ....
LLJ..MM

Gerakan 115: G-down
AA.CD..
..BCDF.
G.BPPFK
GHIIIF.
GHJ....
LLJ..MM

Gerakan 116: M-left
AA.CD..
..BCDF.
G.BPPFK
GHIIIF.
GHJ....
LLJ.MM.

Gerakan 117: M-left
AA.CD..
..BCDF.
G.BPPFK
GHIIIF.

GHJ....
LLJMM..

Gerakan 118: F-down
AA.CD..
..BCD..
G.BPPFK
GHIIIF.
GHJ..F.
LLJMM..

Gerakan 119: F-down
AA.CD..
..BCD..
G.BPP.K
GHIIIF.
GHJ..F.
LLJMMF.

Gerakan 120: P-right-to-exit
AA.CD..
..BCD..
G.B..PK
GHIIIF.
GHJ..F.
LLJMMF.

Test Case 2 distance

Input
6 6 11 AAB..F G.BCDF KGPPCDF GHJIII GHJ... LL.MM.
Output
No solution found

Test Case 3 distance

Input
6 6 1 ..PP..K .B.... .B.... ....
Output
Solution found! Number of moves: 1 Nodes visited: 2 Execution time: 5 ms Papan Awal ..PP..K .B..... .B..... .... Gerakan 1: P-right-to-exit .....PK .B..... .B..... ....

Test Case 4 distance

Input
6 6 7 AABCCD ..B..D ..BPPEK .....E ..FF..

..GG..

Output
Solution found! Number of moves: 2 Nodes visited: 3 Execution time: 6 ms Papan Awal AABCCD. ..B..D. ..BPPEK .....E. ..FF... ..GG... Gerakan 1: E-down AABCCD. ..B..D. ..BPP.K .....E. ..FF.E. ..GG... Gerakan 2: P-right-to-exit AABCCD. ..B..D. ..B..PK .....E. ..FF.E. ..GG...

Test Case 5 distance

Input
4 5 6 K FAAAA F.BBP F.EEP SS.GG

Output
Solution found! Number of moves: 3 Nodes visited: 10 Execution time: 6 ms Papan Awal ....K FAAAA F.BBP F.EEP SS.GG ..... Gerakan 1: S-right ....K FAAAA F.BBP F.EEP .SSGG ..... Gerakan 2: F-down ....K .AAAA F.BBP F.EEP FSSGG ..... Gerakan 3: A-left ....K AAAA. F.BBP F.EEP FSSGG .....

Test Case 6 distance

Input

```
6 6
9
.AABCC
.DDBE.
.FPPE.K
.F....
..GHHH
IIG...
```

Output

```
Solution found!
Number of moves: 3
Nodes visited: 62
Execution time: 13 ms

Papan Awal
.AABCC.
.DDBE..
.FPPE.K
.F.....
..GHHH.
IIG....

Gerakan 1: C-right
.AAB.CC
.DDBE..
.FPPE.K
.F.....
..GHHH.
IIG....

Gerakan 2: E-up
.AABECC
.DDBE..
.FPP..K
.F.....
..GHHH.
IIG....

Gerakan 3: P-right-to-exit
.AABECC
.DDBE..
.F...PK
```

.F.....
..GHHH.
IIG....

Test Case 7 distance

Input
10 10 15 AAB....F.. ..B.CCCF.M D.B..T..OM D..EET..OM D.PP.T..O.K D..LLL...H .....GGGH J...NNN..H JIII..... J.....
Output
Solution found! Number of moves: 14 Nodes visited: 661 Execution time: 137 ms Papan Awal AAB....F... ..B.CCCF.M. D.B..T..OM. D..EET..OM. D.PP.T..O.K D..LLL...H. .....GGGH. J...NNN..H. JIII..... J..... Gerakan 1: O-up AAB....F... ..B.CCCFOM. D.B..T..OM. D..EET..OM.

D.PP.T....K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.....

Gerakan 2: P-right
AAB....F...
..B.CCCFOM.
D.B..T..OM.
D..EET..OM.
D..PPT....K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.....

Gerakan 3: B-down
AA.....F...
..B.CCCFOM.
D.B..T..OM.
D.BEET..OM.
D..PPT....K
D..LLL...H.
.....GGGH.
J...NNN..H.
JIII.....
J.....

Gerakan 4: G-left
AA.....F...
..B.CCCFOM.
D.B..T..OM.
D.BEET..OM.
D..PPT....K
D..LLL...H.
.....GGG.H.
J...NNN..H.
JIII.....
J.....

Gerakan 5: O-up
AA.....FO..
..B.CCCFOM.

D.B..T..OM.
D.BEET...M.
D..PPT....K
D..LLL...H.
.....GGG.H.
J...NNN..H.
JIII.......
J.........

Gerakan 6: N-right
AA.....FO..
..B.CCCFOM.
D.B..T..OM.
D.BEET...M.
D..PPT....K
D..LLL...H.
.....GGG.H.
J....NNN.H.
JIII.......
J.........

Gerakan 7: H-down
AA.....FO..
..B.CCCFOM.
D.B..T..OM.
D.BEET...M.
D..PPT....K
D..LLL.....
.....GGG.H.
J....NNN.H.
JIII.....H.
J.........

Gerakan 8: B-down
AA.....FO..
....CCCFOM.
D.B..T..OM.
D.BEET...M.
D.BPPT....K
D..LLL.....
.....GGG.H.
J....NNN.H.
JIII.....H.
J.........

Gerakan 9: J-up

AA.....FO..
....CCCFOM.
D.B..T..OM.
D.BEET...M.
D.BPPT....K
D..LLL.....
J....GGG.H.
J....NNN.H.
JIII.....H.
.....

Gerakan 10: B-down
AA.....FO..
....CCCFOM.
D....T..OM.
D.BEET...M.
D.BPPT....K
D.BLLL.....
J....GGG.H.
J....NNN.H.
JIII.....H.
.....

Gerakan 11: C-left
AA.....FO..
...CCC.FOM.
D....T..OM.
D.BEET...M.
D.BPPT....K
D.BLLL.....
J....GGG.H.
J....NNN.H.
JIII.....H.
.....

Gerakan 12: C-left
AA.....FO..
..CCC..FOM.
D....T..OM.
D.BEET...M.
D.BPPT....K
D.BLLL.....
J....GGG.H.
J....NNN.H.
JIII.....H.
.....

Gerakan 13: T-up
AA.....FO..
..CCCT.FOM.
D....T..OM.
D.BEET...M.
D.BPP.....K
D.BLLL.....
J....GGG.H.
J....NNN.H.
JIII.....H.
.....

Gerakan 14: P-right-to-exit
AA.....FO..
..CCCT.FOM.
D....T..OM.
D.BEET...M.
D.B......PK
D.BLLL.....
J....GGG.H.
J....NNN.H.
JIII.....H.
.....

Test Case 8 distance

Input
.7 7 9 KKK.... RRRRQ.. K..NQMPP W.NLLM. W.X..M. ..X.... ..OOO..
Output
pintu keluar K tidak valid, ditemukan: 3

Test Case 9 distance

Input
5 5 6   K A.ABB A.CCC DDDF. ...FP .EEEP
Output
Solution found! Number of moves: 2 Nodes visited: 3 Execution time: 5 ms Papan Awal ....K A..BB A.CCC DDDF. ...FP .EEEP Gerakan 1: B-left ....K A.BB. A.CCC DDDF. ...FP .EEEP Gerakan 2: C-left ....K A.BB. ACCC. DDDF. ...FP .EEEP

3.4 Analisis Hasil Pengujian

Uniform Cost Search (UCS) menunjukkan performa yang optimal dari segi hasil solusi, namun memerlukan waktu dan eksplorasi node yang sangat besar terutama pada papan kompleks. Misalnya, pada *Test Case 1*, UCS berhasil menemukan solusi dalam 7 langkah, namun harus mengeksplorasi 38.496 node dengan waktu eksekusi mencapai 1394 ms. Di sisi lain, UCS juga mampu menyelesaikan *Test Case 3* hanya dalam 1 langkah dengan 6 node dikunjungi dan waktu 4 ms. Pada kasus seperti *Test Case 4* dan *Test Case 6*, UCS tetap efisien secara langkah (2–3 langkah), namun tetap memproses puluhan hingga ratusan node, menunjukkan kurangnya efisiensi pada pencarian.

A* dengan heuristik blocking memberikan hasil yang sangat baik dalam efisiensi eksplorasi dibandingkan UCS. Pada *Test Case 1*, A* hanya perlu mengunjungi 227 node dan menyelesaikan dalam waktu 31 ms, jauh lebih efisien daripada UCS. Bahkan untuk kasus besar seperti *Test Case 7*, A* masih dapat menyelesaikannya dengan 8 langkah dan 41.368 node dikunjungi dalam 7518 ms. Meskipun tidak seefisien A* distance, algoritma ini memberikan performa baik pada *Test Case 4*, *Test Case 5*, dan *Test Case 6*, dengan node yang dikunjungi jauh lebih sedikit daripada UCS (masing-masing 7, 16, dan 11 node).

Greedy Best First Search (GBFS) memiliki kinerja tercepat dalam beberapa kasus namun tidak menjamin solusi optimal. Pada *Test Case 1*, GBFS berhasil menemukan solusi namun dengan langkah yang sangat panjang yaitu 47 langkah dan 1381 node dikunjungi, jauh lebih banyak dari UCS atau A*. Hal ini menunjukkan bahwa meskipun cepat, jalur yang diambil cenderung tidak efisien. Bahkan pada *Test Case 6*, jumlah langkah mencapai 114 langkah, menandakan heuristik GBFS belum cukup akurat. Di sisi lain, untuk kasus sederhana seperti *Test Case 3* atau *Test Case 4*, GBFS tetap dapat menemukan solusi cepat dengan langkah pendek dan jumlah node yang relatif sedikit.

A* dengan distance heuristic menjadi algoritma yang paling efisien di antara semua. Pada *Test Case 1*, A* distance hanya perlu 105 node dan menyelesaikan dalam 16 ms dengan langkah solusi sebanyak 9. Pada *Test Case 6*, hanya dengan 3 langkah, A* distance berhasil menemukan solusi dengan 6 node dikunjungi dalam 5 ms. Hal yang sama terjadi di *Test Case 3* dan *Test Case 4*, menunjukkan performa sangat baik untuk puzzle sederhana. Pada *Test Case 7* yang kompleks, algoritma ini juga menunjukkan keunggulan dengan menyelesaikan dalam 14 langkah, 661 node, dan waktu 137 ms, jauh lebih efisien dibanding A* blocking dan UCS. Secara umum, A* distance unggul dalam hal jumlah node yang dikunjungi, waktu eksekusi, serta optimalitas langkah, menjadikannya pilihan terbaik di antara keempat algoritma.

Dalam ranah pencarian jalur (*pathfinding*), algoritma seperti UCS (Uniform Cost Search), A*, dan Greedy Best First Search dibangun di atas prinsip traversal graf dengan tujuan mencapai simpul akhir seefisien mungkin. UCS merupakan algoritma *uninformed search* yang menjelajahi node berdasarkan *path cost* terkecil dari titik awal. Hal ini sesuai dengan teori bahwa UCS akan menghasilkan solusi pasti optimal, tetapi tidak efisien dari sisi eksplorasi ruang karena tidak

memiliki panduan arah ke tujuan. Hasilnya dapat dilihat dari *Test Case 1*, di mana UCS memang menemukan solusi optimal dalam 7 langkah, namun harus mengeksplorasi 38.496 node dengan waktu 1394 ms, cerminan langsung dari eksplorasi menyeluruh (*exhaustive*) yang menjadi karakteristik UCS.

Dengan demikian, hasil pengujian memperkuat teori bahwa pemilihan heuristik sangat menentukan performa dalam *informed search algorithms*, dan bahwa meskipun semua algoritma bisa menyelesaikan masalah, tidak semua melakukannya secara efisien atau optimal. Untuk sistem nyata yang menuntut respons cepat dan akurasi tinggi, A* dengan heuristik yang dirancang secara tepat adalah pendekatan yang paling disarankan.

3.5 Analisis Algoritma

Apakah heuristic pada A* admissible?

Heuristik yang digunakan, yaitu jumlah mobil penghalang di jalur keluar dan jarak ke pintu keluar, adalah admissible karena nilainya tidak pernah melebih-lebihkan cost aktual ke solusi. Dengan demikian, heuristik ini menjamin A* tetap optimal.

Apakah UCS sama dengan BFS pada Rush Hour?

Secara struktur eksplorasi, UCS mirip dengan BFS jika semua langkah memiliki cost yang sama. Namun, UCS memprioritaskan berdasarkan g(n), bukan urutan kedatangan simpul. Dalam kasus Rush Hour, jika tiap langkah diberi bobot sama, maka jalur dan urutan eksplorasi UCS bisa identik dengan BFS.

Apakah A* lebih efisien dibanding UCS?

Secara teoritis, ya. A* menggunakan informasi tambahan berupa heuristik yang mengarahkan pencarian ke arah solusi, sehingga ruang pencarian lebih kecil dibanding UCS yang menyebar ke semua arah. Hal ini ditunjukkan dari jumlah node yang dikunjungi pada eksperimen: UCS sering kali mengunjungi puluhan ribu node, sedangkan A* hanya ratusan.

Apakah GBFS menjamin solusi optimal?

Tidak. GBFS hanya mempertimbangkan h(n) dan tidak memperhatikan g(n). Hal ini menyebabkan algoritma bisa memilih jalur yang terlihat menjanjikan tapi ternyata memiliki cost lebih besar. Contoh pada pengujian menunjukkan GBFS menghasilkan 47 langkah, sementara A* hanya 7 langkah pada kasus yang sama.

3.6 Implementasi Bonus

Terdapat dua heuristic tambahan dalam program:

- blockingCarsHeuristic (jumlah penghalang di depan mobil utama)
- distanceAndBlockingHeuristic (gabungan jarak + penghalang dengan bobot 2x)

blockingCarsHeuristic, yaitu heuristik yang menghitung jumlah mobil yang menghalangi jalur keluar primary piece. Heuristik ini bekerja dengan mengecek baris atau kolom yang dilalui oleh mobil utama hingga ke pintu keluar dan menghitung berapa banyak sel yang ditempati oleh mobil lain (bukan sel kosong '.' atau pintu keluar 'K'). Kedua adalah distanceAndBlockingHeuristic, yang merupakan penggabungan antara jarak Manhattan mobil utama ke pintu keluar (distance) dan jumlah penghalang (blockingCars) yang dikalikan dengan bobot tertentu (dalam hal ini 2 * blockingCars). Heuristik gabungan ini dirancang untuk memberikan estimasi biaya yang lebih realistis, karena tidak hanya memperhitungkan seberapa jauh goal dicapai, tetapi juga mempertimbangkan hambatan langsung di jalur solusi. Kedua heuristik ini bersifat *admissible* karena tidak melebih-lebihkan biaya sesungguhnya untuk mencapai solusi, sehingga tetap menjamin optimalitas pada A*. Dalam implementasinya, heuristik-heuristik ini diterapkan sebagai parameter fungsi Function<Board, Integer> pada konstruktor A*, yang memungkinkan pengguna memilih jenis heuristic saat menjalankan program.

BAB IV
LAMPIRAN

Repository Github: https://github.com/iannn23/Tucil3_13523134.git

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	✓	

Daftar Pustaka:

https://www.trivusi.web.id/2022/10/apa-itu-algoritma-uniform-cost-search.html

https://www.geeksforgeeks.org/uniform-cost-search-ucs-in-ai/

https://www.graphable.ai/blog/pathfinding-algorithms/

https://socjs.telkomuniversity.ac.id/ojs/index.php/indojc/article/view/677?__cf_chl_tk=jtf1_5fn8sazQB_y.jdHwUwtuV.meS4RjaW.UkUzHcM-1747792481-1.0.1.1-nff9OjzpYqUTUmZbi8NPOfXKnBpqbOLf74VGd.54ahI

https://www.trivusi.web.id/2023/01/algoritma-a-star.html

https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/