# LAPORAN TUGAS KECIL III

# IF2211 - STRATEGI ALGORITMA

## Kompresi Gambar Dengan Metode Quadtree


**Disusun oleh :**

Muhammad Alfansya - 13523005

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

**2025**

# Daftar Isi

# BAB I

# Penjelasan Algoritma

## 1.1. Uniform Cost Search (UCS)

Uniform Cost Search (UCS) merupakan algoritma uninformed pathfinding, yang berarti UCS tidak menggunakan heuristik mengenai jarak ke tujuan akhir. UCS menggunakan graf berbobot untuk menemukan jalur dengan biaya terendah dari simpul awal hingga tujuan akhir. UCS menggunakan priority queue untuk menyimpan node, node dengan biaya kumulatif terendah akan dijelajahi terlebih dahulu. UCS menggunakan fungsi evaluasi f(n) = g(n), di mana g(n) adalah biaya kumulatif dari awal ke node n. Karena UCS mengeksplorasi node dengan biaya paling murah terlebih dahulu, algoritma ini menjamin solusi yang ditemukan adalah solusi dengan total biaya minimum,

Algoritma UCS

Langkah proses algoritma UCS :

1. Inisialisasi node root, ditambahkan ke priority queue dengan biaya kumulatif 0
2. Node dengan biaya kumulatif terendah dikeluarkan dari priority queue dan node tersebut dijelajahi
3. Untuk setiap tetangga dari node yang dikunjungi, hitung total cost dari node awal hingga node saat ini, jika node tidak ada di priority queue, tambahkan, jika sudah ada, perbarui cost nya
4. Periksa apakah sudah mencapai node tujuan, jika sudah, algoritma mengembalikan total cost dan jalur yang dikunjungi
5. Ulangi hingga priority queue kosong atau tujuan tercapai

```
Procedure UCS(start, goal)

priorityqueue = [0, start]
Visited = boolean[]

While priorityqueue do
```

```
Curr_cost, curr_node = priorityqueue.deque()

If (curr_node == goal) then return current_cost,
      path_to_current_node //solution found

For (neighbor, cost in curr_node) do
      Total_cost ← total_cost + curr_cost
      If (total_cost < neighbor.cost && neighbor not in
      visited) then
            Neighbor.cost ← total_cost
            Neighbor.node ← curr_node
            priorityqueue.queue (total_cost, neighbor)
            Visited ← neighbor
```

## 1.2. Greedy Best First Search

Greedy Best First Search adalah algoritma yang mencari path paling menjanjikan dari titik awal hingga ke titik tujuan. Algoritma ini memprioritaskan jalur yang terlihat paling menjanjikan meskipun bukan jalur terdekat sebenarnya. Algoritma ini bekerja dengan mengevaluasi biaya dari setiap jalur yang mungkin dan kemudian menjelajahi jalur dengan biaya terendah Algoritma ini menggunakan fungsi evaluasi f(n) yang hanya bergantung pada estimasi menuju tujuan, yaitu f(n) = h(n), di mana h(n) adalah nilai heuristik dari suatu simpul.

Tahapan eksekusi GBFS dimulai dari inisialisasi simpul awal yang langsung dimasukkan ke dalam priority queue dengan prioritas berdasarkan nilai heuristiknya. Selanjutnya, algoritma secara iteratif mengambil simpul dengan nilai heuristik terendah dari antrian untuk diperluas. Setiap simpul yang diekspansi akan dievaluasi tetangga-tetangganya, dan untuk setiap tetangga yang belum pernah dikunjungi, algoritma akan menghitung nilai heuristik dan menambahkannya ke dalam priority queue. Setelah simpul diekspansi, GBFS akan memeriksa apakah simpul tersebut merupakan tujuan. Jika ya, maka pencarian berhenti dan jalur menuju tujuan dikembalikan. Proses pencarian terus berulang hingga simpul tujuan ditemukan atau antrian kosong, yang menandakan bahwa tidak ada jalur yang tersedia. Karena pemilihan jalur sepenuhnya bergantung pada estimasi heuristik, GBFS tidak menjamin bahwa solusi yang ditemukan adalah solusi optimal.

```
procedure GBFS(start, goal)

priorityqueue = [0, start]
Visited = boolean[]

While priorityqueue do
   Curr_h, curr_node = priorityqueue.deque()

  If (visited[curr_node]) then continue to next iteration

   If (curr_node == goal) then path_to_current_node

   For (neighbor, cost in curr_node) do
        If ( neighbor not in visited) then
             priorityqueue.queue(neighbor.h, neighbor)
             Visited ← neighbor
```

## 1.3. A*

A* merupakan algoritma pencarian yang menggabungkan keunggulan UCS dan GBFS dengan menggunakan fungsi evaluasi f(n) = g(n) + h(n). A* mempertimbangkan baik biaya yang telah dikeluarkan (g(n)) maupun estimasi biaya ke tujuan (h(n)). Jika heuristik h(n) yang digunakan

bersifat *admissible*—yaitu tidak pernah melebihi biaya minimum sebenarnya dari node ke tujuan—maka A* menjamin solusi optimal.

A* memprioritaskan simpul dengan nilai f(n) terkecil karena memperhitungkan g(n) (total biaya aktual) dan biaya estimasi (h(n)).

Tahapan proses algoritma A* :

1. Inisialisasi simpul awal yang dimasukkan ke priority queue, nilai g awal 0 dan h dihitung dengan fungsi heuristik, simpul ini adalah simpul awal yang diproses.
2. Keluarkan simpul dengan nilai f terkecil dari priorityqueue, jelajah seluruh tetangga dari simpul ini.
3. Untuk setiap tetangga yang dikunjungi, algoritma akan menghitung nilai heuristik dan biaya nyata dari simpul awal ke simpul saat ini, jika simpul tetangga belum pernah dikunjungi, tambahkan ke priority queue
4. algoritma akan memeriksa tiap simpul yang dikunjungi apakah simpul tersebut adalah simpul tujuan. Jika iya, maka algoritma mengembalikan total biaya dan jalur yang ditempuh.
5. Proses diulangi hingga priority queue kosong atau simpul tujuan telah ditemukan

```
procedure GBFS(start, goal)

priorityqueue = [0, start]
Visited = boolean[]

While priorityqueue do
   Curr_h, curr_node = priorityqueue.deque()

  If (visited[curr_node]) then continue to next iteration

   If (curr_node == goal) then path_to_current_node

   For (neighbor, cost in curr_node) do
```

```
        If ( neighbor not in visited) then
             priorityqueue.queue(neighbor.h, neighbor)
             Visited ← neighbor
```

# BAB II
# Analisis Algoritma

## 2.1. Definisi f(n) dan g(n)

Dalam algoritma pencarian jalur, terdapat tiga komponen utama yang mempengaruhi proses pencarian solusi: actual cost (g(n)), estimasi biaya ke tujuan (h(n)), dan total evaluasi biaya (f(n)). Fungsi g(n) mengukur total biaya yang telah dikeluarkan untuk mencapai sebuah simpul n dari simpul awal, dengan menjumlahkan seluruh bobot lintasan yang telah dilalui. h(n) atau fungsi heuristik memberikan perkiraan biaya dari simpul n menuju simpul tujuan. Meskipun perkiraan ini tidak harus akurat, penting bagi h(n) untuk tidak melebih-lebihkan biaya sebenarnya agar tetap memenuhi sifat *admissible*—yaitu memastikan bahwa solusi optimal tetap dapat ditemukan, seperti yang dipersyaratkan dalam algoritma A*. Fungsi f(n) sendiri merupakan gabungan dari g(n) dan h(n), yaitu f(n) = g(n) + h(n), yang digunakan sebagai nilai evaluasi total untuk simpul n.

## 2.2. Apakah heuristik yang digunakan pada Algoritma A* Admissible

Dalam algoritma A*, sebuah heuristik dikatakan *admissible* jika nilai perkiraannya terhadap biaya dari simpul n ke tujuan tidak pernah melebihi biaya aktual dari jalur terpendek. Secara formal, heuristik h(n) memenuhi sifat *admissible* apabila untuk setiap simpul n, berlaku $h(n) \leq h^*(n)$, dengan $h^*(n)$ merupakan biaya sesungguhnya dari simpul n ke tujuan. Sifat ini sangat penting karena menjamin bahwa A* akan selalu menemukan solusi yang optimal. Jika nilai heuristik terlalu tinggi (overestimate), maka algoritma A* dapat melewatkan jalur terbaik.

## 2.3. Algoritma UCS dan BFS pada Rush Hour

Secara umum, algoritma Uniform Cost Search (UCS) dan Breadth-First Search (BFS) memiliki struktur kerja yang serupa, terutama jika seluruh aksi dalam ruang pencarian memiliki biaya yang sama. Dalam situasi seperti permainan *Rush Hour*, apabila setiap pergerakan kendaraan dihitung sebagai satu langkah dengan bobot yang seragam, maka UCS dan BFS akan menelusuri node dalam urutan yang sama dan menghasilkan solusi yang identik. Hal ini terjadi karena dalam kasus seperti itu, kedalaman pencarian pada BFS sebanding dengan akumulasi biaya g(n) pada UCS.

Namun demikian, terdapat perbedaan prinsip mendasar antara keduanya. BFS mengembangkan node berdasarkan tingkat kedalaman dari node awal, tanpa mempertimbangkan besar kecilnya biaya antar node. Di sisi lain, UCS memprioritaskan node dengan total biaya terkecil dari awal (g(n)), sehingga lebih sensitif terhadap variasi bobot antar langkah.

## 2.4. Efisiensi A* dan UCS pada Rush Hour

Penerapan algoritma A* dan Uniform Cost Search (UCS) pada penyelesaian puzzle *Rush Hour* memperlihatkan perbedaan mencolok dalam hal efisiensi eksplorasi simpul, waktu penyelesaian, serta kompleksitas pencarian. A* umumnya lebih efisien karena memanfaatkan kombinasi antara biaya aktual dari simpul awal ke simpul saat ini (g(n)) dan estimasi biaya menuju tujuan (h(n)). Pendekatan ini membuat A* lebih terarah, sehingga mampu menghindari eksplorasi jalur yang tidak relevan dan mengurangi jumlah simpul yang diproses.

Sebaliknya, UCS hanya mempertimbangkan biaya aktual tanpa bantuan estimasi ke tujuan, sehingga pencariannya cenderung menjelajahi lebih banyak node, termasuk jalur yang tidak mengarah langsung pada solusi. Akibatnya, UCS bisa menghabiskan lebih banyak waktu untuk menyelesaikan masalah, terutama pada ruang pencarian yang luas.

## 2.5. Solusi Optimal oleh Algoritma Greedy Best First Search

Algoritma Greedy Best-First Search (GBFS) merupakan pendekatan pencarian yang hanya berfokus pada estimasi jarak ke tujuan (h(n)) tanpa mempertimbangkan biaya aktual yang telah dikeluarkan dari titik awal (g(n)). Karena hanya mengandalkan nilai heuristik untuk memilih node berikutnya, GBFS bersifat *greedy*, cenderung langsung mengejar jalur yang tampak paling dekat ke tujuan, walaupun belum tentu jalur tersebut merupakan yang tercepat atau paling efisien secara keseluruhan.

Dalam konteks penyelesaian *Rush Hour*, GBFS dapat bekerja dengan cepat karena menghindari eksplorasi luas yang tidak relevan, namun hasilnya tidak selalu optimal. GBFS bisa saja memilih jalur yang lebih panjang atau bahkan buntu jika estimasi heuristiknya menyesatkan. Selain itu, jika tidak ada mekanisme untuk mencatat node yang telah dikunjungi, algoritma ini bahkan bisa gagal menemukan solusi, karena berpotensi mengeksplorasi ulang node yang sama atau terjebak dalam siklus.

# BAB III

## Source Code Program

Main.java

```
import java.util.Scanner;

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



        System.out.print("Enter the path to the puzzle file: ");
        String filePath = scanner.nextLine();



        System.out.println("Select algorithm:");
        System.out.println("1. UCS (Uniform Cost Search)");
        System.out.println("2. Greedy Best First Search");
        System.out.println("3. A* (A-Star)");
        System.out.print("Enter your choice (1-3): ");
        int algorithmChoice = scanner.nextInt();

        try {

            Board puzzle = ioHandler.readPuzzleFromFile(filePath);

            System.out.println("Papan Awal:");
            puzzle.printBoard(null, ' ', ' ');
            System.out.println("Posisi Exit (K): (" + puzzle.getExitPosition()[0] + ", " +
puzzle.getExitPosition()[1] + ")");



            switch (algorithmChoice) {
                case 1:
                    Solver.UCS(puzzle);
                    break;
```

```
                case 2:
                    Solver.GBFS(puzzle);
                    break;
                case 3:

                    Solver.AStar(puzzle);
                    break;
                default:
                    System.out.println("Invalid choice. Using UCS by default.");
                    Solver.UCS(puzzle);
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
            e.printStackTrace();
        } finally {
            scanner.close();
        }
    }
}
```

Board.java

```
import java.util.*;

public class Board {
    private int rows;
    private int cols;
    private char[][] board;
    private int[] exitPosition;
    private int[] primaryPosition;
    private boolean IsPrimaryHorizontal;

    public Board(int rows, int cols, char[][] board) {
        this(rows, cols, board, null);
```

```
    }

    public Board(int rows, int cols, char[][] board, int[] exitPos) {
        this.rows = rows;
        this.cols = cols;
        this.board = new char[rows][cols];

        // Copy the board
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                this.board[i][j] = board[i][j];

                // Find the exit position ('K') if not provided
                if (exitPos == null && board[i][j] == 'K') {
                    this.exitPosition = new int[]{i, j};
                }

                // Find the primary piece position ('P')
                if (board[i][j] == 'P' && primaryPosition == null) {
                    primaryPosition = new int[]{i, j};
                }
            }
        }

        // Use provided exit position if available
        if (exitPos != null) {
            this.exitPosition = new int[]{exitPos[0], exitPos[1]};
        }

        // Verify exitPosition is set
        if (this.exitPosition == null) {
            System.err.println("Warning: Exit position (K) not found in the puzzle!");
        }

        // Determine if the primary piece is horizontal or vertical
        determineOrientationOfPrimaryPiece();
    }
```

```java
    public Board(Board other) {
        this.rows = other.rows;
        this.cols = other.cols;
        this.board = new char[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                this.board[i][j] = other.board[i][j];
            }
        }

        this.exitPosition = (other.exitPosition != null) ? new
int[]{other.exitPosition[0], other.exitPosition[1]} : null;
        this.primaryPosition = (other.primaryPosition != null) ?
                                    new int[]{other.primaryPosition[0],
other.primaryPosition[1]} : null;
        this.IsPrimaryHorizontal = other.IsPrimaryHorizontal;
    }

    // Getter for exitPosition
    public int[] getExitPosition() {
        return exitPosition;
    }

    public int getRows() {
        return rows;
    }

    public int getCols() {
        return cols;
    }

    public char[][] getBoard() {
        return board;
    }

    public boolean isPrimaryHorizontal() {
        return IsPrimaryHorizontal;
```

```
    }

    public void setPrimaryPosition(int[] position) {
        this.primaryPosition = position;
    }


    private void determineOrientationOfPrimaryPiece() {
        // Check if there's a primary piece to the right of the found position
        if (primaryPosition != null) {
            if (primaryPosition[1] + 1 < cols &&
                board[primaryPosition[0]][primaryPosition[1] + 1] == 'P') {
                IsPrimaryHorizontal = true;
            } else if (primaryPosition[0] + 1 < rows &&
                       board[primaryPosition[0] + 1][primaryPosition[1]] == 'P') {
                IsPrimaryHorizontal = false;
            } else {
                // Single cell piece, defaulting to horizontal
                IsPrimaryHorizontal = true;
            }

        } else {
            System.err.println("Warning: Primary piece (P) not found in the puzzle!");
        }
    }

    public boolean isSolved() {
        if (exitPosition == null || primaryPosition == null) {
            return false;
        }

        // Find all positions of the primary piece
        List<int[]> primaryPiecePositions = new ArrayList<>();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (board[i][j] == 'P') {
                    primaryPiecePositions.add(new int[]{i, j});
                }
```

```
            }
        }

        if (primaryPiecePositions.isEmpty()) {
            return true;
        }

        if (IsPrimaryHorizontal) {
            primaryPiecePositions.sort(Comparator.comparingInt(pos -> pos[1]));

            int[] leftmost = primaryPiecePositions.get(0);
            int[] rightmost = primaryPiecePositions.get(primaryPiecePositions.size() -
1);

            // jika exit ada di kiri
            if (exitPosition[1] == -1) {
                return exitPosition[0] == leftmost[0] && rightmost[1] == -1;
            }
            else{
                return exitPosition[0] == rightmost[0] && leftmost[1] == cols;
            }


        } else {// Vertical
            primaryPiecePositions.sort(Comparator.comparingInt(pos -> pos[0]));

            // Get bottommost position of the primary piece
            int[] topmost = primaryPiecePositions.get(0);
            int[] bottommost = primaryPiecePositions.get(primaryPiecePositions.size() -
1);

            // jika exit ada di atas
            if (exitPosition[0] == -1) {
                return exitPosition[1] == topmost[1] && bottommost[0] == -1;
            }
            else{ // exit di bawah
                return exitPosition[1] == bottommost[1] && topmost[0] == rows;
            }
```

```
        }

    }

    public String getBoardString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                sb.append(board[i][j]);
            }
        }
        return sb.toString();
    }

    public String getOutputString() {
        StringBuilder sb = new StringBuilder();

        if (exitPosition[0] == -1){
            for (int i = 0; i < cols; i++) {
                if (i == exitPosition[1]) {
                    sb.append("K");
                } else {
                    sb.append("  ");
                }
            }
            sb.append("\n");
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    sb.append(board[i][j]);
                    sb.append(" ");
                }
                sb.append("\n");
            }
        } else if (exitPosition[1] == -1) {
            for (int i = 0; i < rows; i++) {
                if (i == exitPosition[0]) {
                    sb.append("K ");
                    for (int j = 0; j < cols; j++) {
```

```
                    sb.append(board[i][j] + " ");
                }
            } else {
                sb.append("  ");
                for (int j = 0; j < cols; j++) {
                    sb.append(board[i][j] + " ");
                }
            }
            sb.append("\n");
        }
    } else if (exitPosition[0] == rows) {
        for (int i = 0; i < rows+1; i++) {
            if (i == exitPosition[0]) {
                for (int j = 0; j < cols; j++) {
                    if (j == exitPosition[1]) {
                        sb.append("K ");
                    } else {
                        sb.append("  ");
                    }
                }
                sb.append("\n");
            } else {
                for (int j = 0; j < cols; j++) {
                    sb.append(board[i][j] + " ");
                }
                sb.append("\n");
            }
        }
    } else {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                sb.append(board[i][j] + " ");
            }
            if (i == exitPosition[0]) {
                sb.append("K");
            }
            sb.append("\n");
        }
```

```
        }

        return sb.toString();
    }

    public void printBoard(int[] previousPosition, char movedPiece, char direction) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                char cell = board[i][j];

                // ANSI color codes for console output
                String primaryPieceColor = "\u001B[31m"; // Red
                String exitColor = "\u001B[32m"; // Green
                String movedPieceColor = "\u001B[33m"; // Yellow
                String resetColor = "\u001B[0m";

                // Determine if this position corresponds to the moved piece
                boolean isMovedPiece = false;
                if (previousPosition != null && movedPiece != ' ') {
                    if (cell == movedPiece) {
                        isMovedPiece = true;
                    }
                }

                // Apply colors based on cell content
                if (cell == 'P') {
                    System.out.print(primaryPieceColor + cell + resetColor);
                } else if (cell == 'K') {
                    System.out.print(exitColor + cell + resetColor);
                } else if (isMovedPiece) {
                    System.out.print(movedPieceColor + cell + resetColor);
                } else {
                    System.out.print(cell);
                }
                System.out.print(" ");
            }
            System.out.println();
        }
```

```
    }
}
```

State.java

```
import java.util.*;

class State {
    Board board;
    int cost;
    State parent;
    int[] previousPosition;
    char movedPiece;
    char direction;

    public State(Board board, int cost, State parent,
                 int[] previousPosition, char movedPiece, char direction) {
        this.board = board;
        this.cost = cost;
        this.parent = parent;
        this.previousPosition = previousPosition;
        this.movedPiece = movedPiece;
        this.direction = direction;
    }

    public State getParent() {
        return parent;
    }

    public List<State> getNextStates(State currentState) {
        List<State> nextStates = new ArrayList<>();
        Map<Character, List<int[]>> piecePositions = new HashMap<>();
        int rows = board.getRows();
```

```
        int cols = board.getCols();
        char[][] boardGrid = board.getBoard();

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                char piece = boardGrid[i][j];
                if (piece != '.' && piece != 'K') {
                    piecePositions.computeIfAbsent(piece, k -> new
ArrayList<>())
                                  .add(new int[]{i, j});
                }
            }
        }

        for (Map.Entry<Character, List<int[]>> entry :
piecePositions.entrySet()) {

            char piece = entry.getKey();
            List<int[]> positions = entry.getValue();

            boolean isHorizontal = true;
            if (positions.size() > 1) {
                isHorizontal = positions.get(0)[0] == positions.get(1)[0];
            }

            if (piece == 'P' && board.isPrimaryHorizontal()) {

                positions.sort(Comparator.comparingInt(pos -> pos[1]));

                int leftmostCol = positions.get(0)[1];
                int rightmostCol = positions.get(positions.size() - 1)[1];

                int row = positions.get(0)[0];

                // exit ada di kiri, gerakan P ke kiri
                if (leftmostCol == 0 && board.getExitPosition()[1] == -1) {
                    Board newBoard = new Board(board);
```

```
                    int[] previousRightmost = positions.get(positions.size() -
1);

                    newBoard.getBoard()[row][leftmostCol] = piece;
                    newBoard.getBoard()[row][previousRightmost[1]] = '.';

                    if (piece == 'P') {
                        newBoard.setPrimaryPosition(new int[]{row,
leftmostCol});
                    }

                    nextStates.add(new State(
                        newBoard,
                        currentState.cost + 1,
                        currentState,
                        new int[]{previousRightmost[0], previousRightmost[1]},
                        piece,
                        'L'
                    ));
                }

                // exit ada di kanan, gerakan P ke kanan
                if (rightmostCol == cols-1 && board.getExitPosition()[1] ==
cols) {
                    Board newBoard = new Board(this.board);

                    int[] previousLeftmost = positions.get(0);

                    newBoard.getBoard()[row][rightmostCol] = piece;
                    newBoard.getBoard()[row][previousLeftmost[1]] = '.';

                    if (piece == 'P') {
                        newBoard.setPrimaryPosition(new int[]{row,
rightmostCol});
                    }

                    nextStates.add(new State(
                        newBoard,
```

```
                        currentState.cost + 1,
                        currentState,
                        new int[]{previousLeftmost[0], previousLeftmost[1]},
                        piece,
                        'R'
                    ));
                }
            }
            else if (piece == 'P' && !board.isPrimaryHorizontal()) {
                int topmostRow = positions.get(0)[0];
                int col = positions.get(0)[1];

                // exit ada di atas, gerakan P ke atas
                if (topmostRow == 0 && board.getExitPosition()[0] == -1) {
                    Board newBoard = new Board(this.board);

                    int[] previousBottommost = positions.get(positions.size() -
1);

                    newBoard.getBoard()[topmostRow][col] = piece;
                    newBoard.getBoard()[previousBottommost[0]][col] = '.';

                    if (piece == 'P') {
                        newBoard.setPrimaryPosition(new int[]{topmostRow,
col});
                    }

                    nextStates.add(new State(
                        newBoard,
                        currentState.cost + 1,
                        currentState,
                        new int[]{previousBottommost[0],
previousBottommost[1]},
                        piece,
                        'U'
                    ));
                }
                int bottommostRow = positions.get(positions.size() - 1)[0];
```

```
                // exit ada di bawah, gerakan P ke bawah
                if (bottommostRow == rows - 1 && board.getExitPosition()[0] ==
rows) {

                    Board newBoard = new Board(this.board);

                    int[] previousTopmost = positions.get(0);

                    newBoard.getBoard()[bottommostRow][col] = piece;
                    newBoard.getBoard()[previousTopmost[0]][col] = '.';

                    if (piece == 'P') {
                        newBoard.setPrimaryPosition(new int[]{bottommostRow,
col});
                    }
                    nextStates.add(new State(
                        newBoard,
                        currentState.cost + 1,
                        currentState,
                        new int[]{previousTopmost[0], previousTopmost[1]},
                        piece,
                        'D'
                    ));
                }
            }
            if (isHorizontal) {
                positions.sort(Comparator.comparingInt(pos -> pos[1]));

                int leftmostCol = positions.get(0)[1];
                int row = positions.get(0)[0];

                if (leftmostCol > 0 && boardGrid[row][leftmostCol - 1] == '.')
{

                    Board newBoard = new Board(this.board);
```

```
                    int[] previousRightmost = positions.get(positions.size() -
1);

                    newBoard.getBoard()[row][leftmostCol - 1] = piece;
                    newBoard.getBoard()[row][previousRightmost[1]] = '.';

                    if (piece == 'P') {
                        newBoard.setPrimaryPosition(new int[]{row, leftmostCol
- 1});
                    }

                    nextStates.add(new State(
                        newBoard,
                        currentState.cost + 1,
                        currentState,
                        new int[]{previousRightmost[0], previousRightmost[1]},
                        piece,
                        'L'
                    ));
                }

                int rightmostCol = positions.get(positions.size() - 1)[1];

                if (rightmostCol < cols-1 && boardGrid[row][rightmostCol + 1]
== '.') {

                    Board newBoard = new Board(this.board);

                    int[] previousLeftmost = positions.get(0);

                    newBoard.getBoard()[row][rightmostCol + 1] = piece;
                    newBoard.getBoard()[row][leftmostCol] = '.';

                    if (piece == 'P') {
                         newBoard.setPrimaryPosition(new int[]{row, leftmostCol
+ 1});
                    }
```

```
                    nextStates.add(new State(
                        newBoard,
                        currentState.cost + 1,
                        currentState,
                        new int[]{previousLeftmost[0], previousLeftmost[1]},
                        piece,
                        'R'
                    ));
                }
            } else {
                positions.sort(Comparator.comparingInt(pos -> pos[0]));

                int topmostRow = positions.get(0)[0];
                int col = positions.get(0)[1];

                if (topmostRow > 0 && boardGrid[topmostRow - 1][col] == '.') {

                    Board newBoard = new Board(this.board);

                    int[] previousBottommost = positions.get(positions.size() -
1);


                    newBoard.getBoard()[topmostRow - 1][col] = piece;
                    newBoard.getBoard()[previousBottommost[0]][col] = '.';

                    if (piece == 'P') {
                        newBoard.setPrimaryPosition(new int[]{topmostRow - 1,
col});
                    }

                    nextStates.add(new State(
                        newBoard,
                        currentState.cost + 1,
                        currentState,
                        new int[]{previousBottommost[0],
previousBottommost[1]},
                        piece,
                        'U'
```

```
                    ));
                }

                int bottommostRow = positions.get(positions.size() - 1)[0];

                if (bottommostRow < rows - 1 && boardGrid[bottommostRow +
1][col] == '.') {

                    Board newBoard = new Board(this.board);

                    int[] previousTopmost = positions.get(0);

                    newBoard.getBoard()[bottommostRow + 1][col] = piece;
                    newBoard.getBoard()[topmostRow][col] = '.';

                    if (piece == 'P') {
                        newBoard.setPrimaryPosition(new int[]{topmostRow + 1,
col});
                    }

                    nextStates.add(new State(
                        newBoard,
                        currentState.cost + 1,
                        currentState,
                        new int[]{previousTopmost[0], previousTopmost[1]},
                        piece,
                        'D'
                    ));
                }
            }
        }
        return nextStates;
    }

    public int getHeuristicCost() {

        int rows = board.getRows();
        int cols = board.getCols();
```

```
        char[][] boardGrid = board.getBoard();

        List<int[]> primaryPiecePositions = new ArrayList<>();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (boardGrid[i][j] == 'P') {
                    primaryPiecePositions.add(new int[]{i, j});
                }
            }
        }

        if (primaryPiecePositions.isEmpty()) {
            return 0;
        }

        if (board.isPrimaryHorizontal()) {
            primaryPiecePositions.sort(Comparator.comparingInt(pos -> pos[1]));

            int[] leftmost = primaryPiecePositions.get(0);
            int[] rightmost =
primaryPiecePositions.get(primaryPiecePositions.size() - 1);

            if (board.getExitPosition()[1] == -1) {
                return leftmost[1];
            } else {
                return cols - 1 - rightmost[1];
            }
        } else {
            primaryPiecePositions.sort(Comparator.comparingInt(pos -> pos[0]));

            int[] topmost = primaryPiecePositions.get(0);
            int[] bottommost =
primaryPiecePositions.get(primaryPiecePositions.size() - 1);

            if (board.getExitPosition()[0] == -1) {
                return topmost[0];
            } else {
                return rows - 1 - bottommost[0];
```

```
            }
        }
    }
}
```

Solver.java

```
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;

public class Solver {
    public static void UCS(Board initialboard) {
        long startTime = System.currentTimeMillis();

        PriorityQueue<State> queue = new PriorityQueue<>(
            Comparator.comparingInt(state -> state.cost)
        );

        Set<String> visited = new HashSet<>();

        State initialState = new State(initialboard, 0, null, null, ' ', 'U');
        queue.add(initialState);
        visited.add(initialState.board.getBoardString());

        int nodesVisited = 0;

        System.out.println("Starting search...");
```

```
        while (!queue.isEmpty()) {

            State currentState = queue.poll();
            nodesVisited++;

            if (currentState.board.isSolved()) {
                long endTime = System.currentTimeMillis();

                System.out.println("\nSolution found!");
                System.out.println("Number of moves: " + currentState.cost);
                System.out.println("Nodes visited: " + nodesVisited);
                System.out.println("Time taken: " + (endTime - startTime) + "
ms");

                ioHandler.saveToFile(nodesVisited, System.currentTimeMillis() -
startTime, currentState);
                printSolution(currentState);
                return;
            }

            List<State> nextStates = currentState.getNextStates(currentState);

            for (State nextState : nextStates) {
                if (!visited.contains(nextState.board.getBoardString())) {
                    queue.add(nextState);
                    visited.add(nextState.board.getBoardString());
                }else if (queue.contains(nextState)) {
                    for (State stateInQueue : queue) {
                        if (stateInQueue.equals(nextState) && stateInQueue.cost
> nextState.cost) {
                            queue.remove(stateInQueue);
                            queue.add(nextState);
                            break;
                        }
                    }
                }
            }
        }
```

```
        ioHandler.saveNoSolutionToFile(nodesVisited, System.currentTimeMillis()
- startTime);
        System.out.println("No solution found.");
        System.out.println("Nodes visited: " + nodesVisited);
        System.out.println("Time taken: " + (System.currentTimeMillis() -
startTime) + " ms");
    }

    public static void GBFS(Board initialboard) {
        long startTime = System.currentTimeMillis();

        PriorityQueue<State> queue = new PriorityQueue<>(
            Comparator.comparingInt(state -> state.getHeuristicCost())
        );

        Set<String> visited = new HashSet<>();

        State initialState = new State(initialboard, 0, null, null, ' ', 'U');
        queue.add(initialState);
        visited.add(initialState.board.getBoardString());

        int nodesVisited = 0;

        System.out.println("Starting Greedy Best-First search...");

        while (!queue.isEmpty()) {
            State currentState = queue.poll();
            nodesVisited++;

            if (currentState.board.isSolved()) {
                long endTime = System.currentTimeMillis();

                System.out.println("\nSolution found!");
                System.out.println("Number of moves: " + currentState.cost);
                System.out.println("Nodes visited: " + nodesVisited);
                System.out.println("Time taken: " + (endTime - startTime) + "
ms");
```

```
                ioHandler.saveToFile(nodesVisited, System.currentTimeMillis() -
startTime, currentState);
                printSolution(currentState);
                return;
            }

            List<State> nextStates = currentState.getNextStates(currentState);

            for (State nextState : nextStates) {
                String nextStateString = nextState.board.getBoardString();

                if (!visited.contains(nextStateString)) {
                    queue.add(nextState);
                    visited.add(nextStateString);
                }
            }

        }
        ioHandler.saveNoSolutionToFile(nodesVisited, System.currentTimeMillis()
- startTime);
        System.out.println("No solution found.");
        System.out.println("Nodes visited: " + nodesVisited);
        System.out.println("Time taken: " + (System.currentTimeMillis() -
startTime) + " ms");
    }

    public static void AStar(Board initialBoard) {
        long startTime = System.currentTimeMillis();

        PriorityQueue<State> queue = new PriorityQueue<>(
            Comparator.comparingInt(state -> state.cost +
state.getHeuristicCost())
        );

        Set<String> visited = new HashSet<>();

        State initialState = new State(initialBoard, 0, null, null, ' ', 'U');
        queue.add(initialState);
```

```
        int nodesVisited = 0;

        System.out.println("Starting A* search...");

        while (!queue.isEmpty()) {
            State currentState = queue.poll();
            nodesVisited++;

            String currentStateString = currentState.board.getBoardString();
            if (visited.contains(currentStateString)) {
                continue;
            }

            visited.add(currentStateString);

            if (currentState.board.isSolved()) {
                long endTime = System.currentTimeMillis();

                System.out.println("\nSolution found!");
                System.out.println("Number of moves: " + currentState.cost);
                System.out.println("Nodes visited: " + nodesVisited);
                System.out.println("Time taken: " + (endTime - startTime) + "
ms");

                ioHandler.saveToFile(nodesVisited, System.currentTimeMillis() -
startTime, currentState);
                printSolution(currentState);
                return;
            }

            List<State> nextStates = currentState.getNextStates(currentState);

            for (State nextState : nextStates) {
                String nextStateString = nextState.board.getBoardString();
                if (!visited.contains(nextStateString)) {
                    queue.add(nextState);
                }
```

```
            }

        }
        ioHandler.saveNoSolutionToFile(nodesVisited, System.currentTimeMillis()
- startTime);
        System.out.println("No solution found.");
        System.out.println("Nodes visited: " + nodesVisited);
        System.out.println("Time taken: " + (System.currentTimeMillis() -
startTime) + " ms");
    }

    private static void printSolution(State finalState) {
        List<State> path = new ArrayList<>();
        State currentState = finalState;

        while (currentState != null) {
            path.add(currentState);
            currentState = currentState.parent;
        }

        Collections.reverse(path);

        System.out.println("Papan Awal:");
        path.get(0).board.printBoard(null, ' ', ' ');

        for (int i = 1; i < path.size(); i++) {
            State state = path.get(i);
            System.out.println("Gerakan " + i + ": " + state.movedPiece + "-" +
                               (state.direction == 'U' ? "atas" :
                                state.direction == 'D' ? "bawah" :
                                state.direction == 'L' ? "kiri" : "kanan"));

            state.board.printBoard(state.previousPosition, state.movedPiece,
state.direction);
        }
    }
}
```

ioHandler.java

```java
import java.io.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ioHandler {
    public static Board readPuzzleFromFile(String filePath) throws IOException {
        BufferedReader reader;
        try {
            reader = new BufferedReader(new FileReader("test/" + filePath + ".txt"));
        } catch (FileNotFoundException e) {

            reader = new BufferedReader(new FileReader(filePath));
        }

        String[] dimensions = reader.readLine().split("\\s+");
        int rows = Integer.parseInt(dimensions[0]);
        int cols = Integer.parseInt(dimensions[1]);

        int numPieces = Integer.parseInt(reader.readLine());

        char[][] tboard = new char[rows+1][cols+1];
        for (int x = 0; x < rows + 1; x++) {
            for (int y = 0; y < cols + 1; y++) {
                tboard[x][y] = ' ';
            }
        }
        int[] exitPosition = null;

        String line;
        int i = 0;
        while ((line = reader.readLine()) != null && i < rows + 1) {
            for (int j = 0; j < cols + 1; j++) {
```

```
            if (j < line.length()) {
                tboard[i][j] = line.charAt(j);
                if (tboard[i][j] == 'K') {
                    int exitRow = i, exitCol = j;
                    if (j == 0 && i < cols){
                        exitCol = -1;
                    }
                    if (i == 0 && j < rows){
                        exitRow = -1;
                    }
                    System.out.println(exitCol + " " + exitRow);
                    exitPosition = new int[]{exitRow, exitCol};
                }
            }
        }
        i++;
    }
    reader.close();

    char[][] board = new char[rows][cols];
    if (exitPosition[0] == -1){
        for (i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                board[i][j] = tboard[i+1][j];

            }
        }
    }
    else if (exitPosition[1] == -1){
        for (i = 0; i < rows; i++) {

            if (i == exitPosition[0]){
                for (int j = 0; j < cols; j++) {
                    board[i][j] = tboard[i][j+1];
                }
            }
            else{
                for (int j = 0; j < cols; j++) {
```

```
                    board[i][j] = tboard[i][j];
                }
            }
        }
    }
    else{
        for (i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                board[i][j] = tboard[i][j];
            }
        }
    }

    return new Board(rows, cols, board, exitPosition);
}

public static void saveToFile(int nodeCount, long executionTime, State solution) {
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Masukkan nama file untuk menyimpan solusi:");
    String filename;
    try {
        filename = "test/" + reader.readLine() + ".txt";
    } catch (IOException e) {
        System.err.println("Error reading input: " + e.getMessage());
        return;
    }
    List<State> path = new ArrayList<>();
    State current = solution;
    while (current != null) {
        path.add(current);
        current = current.getParent();
    }
    Collections.reverse(path);

    try (PrintWriter writer = new PrintWriter(filename)) {
        for (int i = 1; i < path.size(); i++) {
            State state = path.get(i);
            writer.println("Gerakan " + i + ": " + state.movedPiece + "-" +
```

```
                                (state.direction == 'U' ? "atas" :
                                    state.direction == 'D' ? "bawah" :
                                    state.direction == 'L' ? "kiri" : "kanan"));

                    writer.println(state.board.getOutputString());
                }
                writer.println("Visited nodes: " + nodeCount);
                writer.println("Execution time: " + executionTime + " ms");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public static void saveNoSolutionToFile(int nodeCount, long executionTime) {
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Masukkan nama file untuk menyimpan solusi:");
            String filename;
            try {
                filename = "test/" + reader.readLine() + ".txt";
            } catch (IOException e) {
                System.err.println("Error reading input: " + e.getMessage());
                return;
            }
            try (PrintWriter writer = new PrintWriter(filename)) {
                writer.println("Tidak ada solusi ditemukan.");
                writer.println("Visited nodes: " + nodeCount);
                writer.println("Execution time: " + executionTime + " ms");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

}
```

# BAB IV

# Hasil Pengujian

## 4.1. Test Case 1

test > test1.txt

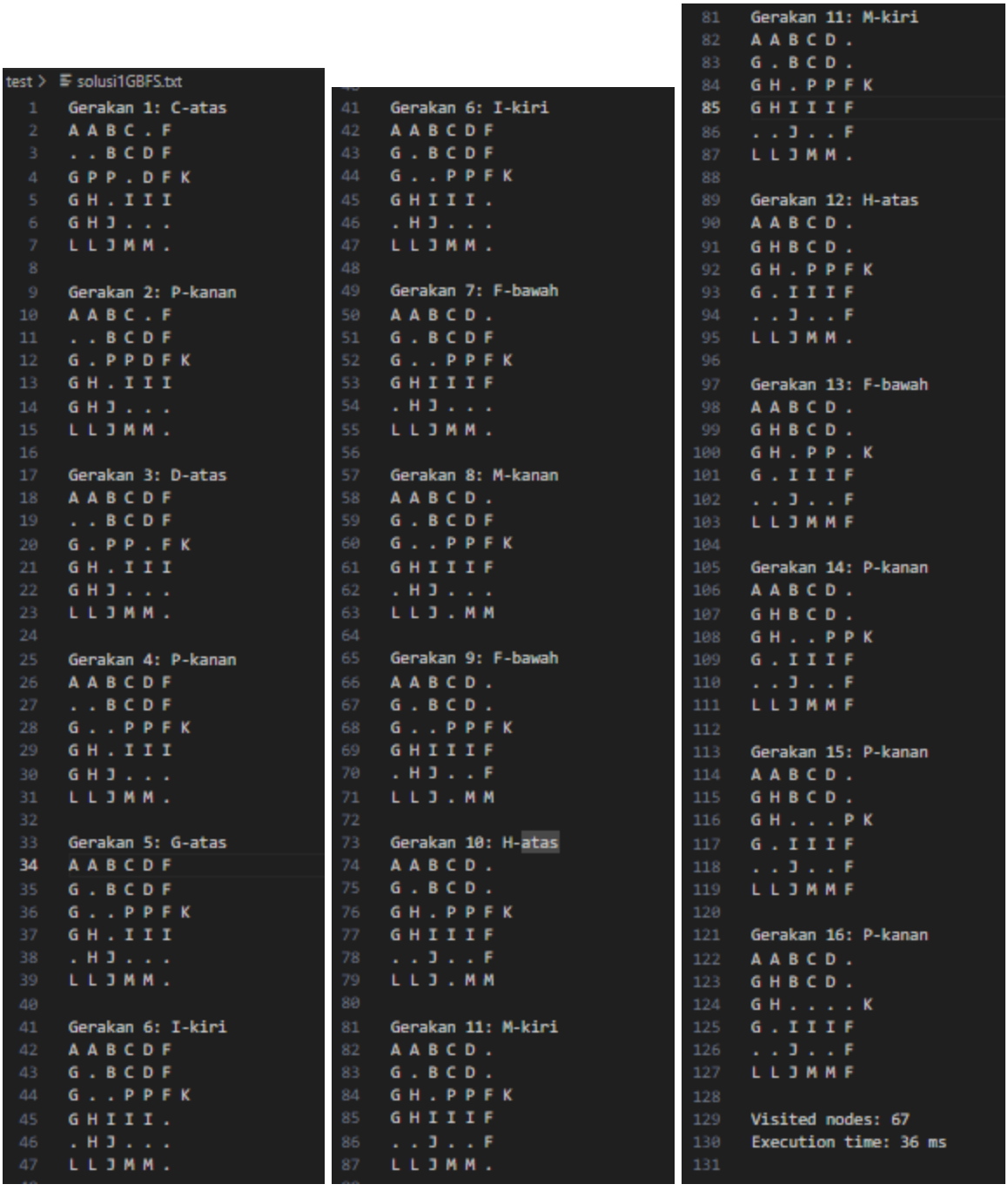
```
1  6 6
2  12
3  AAB..F
4  ..BCDF
5  GPPCDFK
6  GH.III
7  GHJ...
8  LLJMM.
```

**Input, kondisi K di kanan**

test > solusi1UCS.txt

```
Gerakan 1: I-kiri
A A B . . F
. . B C D F
G P P C D F K
G H I I I .
G H J . . .
L L J M M .

Gerakan 2: F-bawah
A A B . . .
. . B C D F
G P P C D F K
G H I I I F
G H J . . .
L L J M M .

Gerakan 3: F-bawah
A A B . . .
. . B C D .
G P P C D F K
G H I I I F
G H J . . F
L L J M M .

Gerakan 4: C-atas
A A B C . .
. . B C D .
G P P . D F K
G H I I I F
G H J . . F
L L J M M .

Gerakan 5: P-kanan
A A B C . .
. . B C D .
G . P P D F K
G H I I I F
G H J . . F
L L J M M .

Gerakan 6: D-atas
A A B C D .
. . B C D .
G . P P . F K
G H I I I F
G H J . . F
L L J M M .

Gerakan 7: F-bawah
A A B C D .
. . B C D .
G . P P . . K
G H I I I F
G H J . . F
L L J M M F

Gerakan 8: P-kanan
A A B C D .
. . B C D .
G . . P P . K
G H I I I F
G H J . . F
L L J M M F

Gerakan 9: P-kanan
A A B C D .
. . B C D .
G . . . P P K
G H I I I F
G H J . . F
L L J M M F

Gerakan 10: P-kanan
A A B C D .
. . B C D .
G . . . . P K
G H I I I F
G H J . . F
L L J M M F

Gerakan 11: P-kanan
A A B C D .
. . B C D .
G . . . . . K
G H I I I F
G H J . . F
L L J M M F

Visited nodes: 907
Execution time: 73 ms
```

**Output dengan UCS**



test > solusi1GBFS.txt

```
Gerakan 1: C-atas
A A B C . F
. . B C D F
G P P . D F K
G H . I I I
G H J . . .
L L J M M .

Gerakan 2: P-kanan
A A B C . F
. . B C D F
G . P P D F K
G H . I I I
G H J . . .
L L J M M .

Gerakan 3: D-atas
A A B C D F
. . B C D F
G . P P . F K
G H . I I I
G H J . . .
L L J M M .

Gerakan 4: P-kanan
A A B C D F
. . B C D F
G . . P P F K
G H . I I I
G H J . . .
L L J M M .

Gerakan 5: G-atas
A A B C D F
G . B C D F
G . . P P F K
G H . I I I
. H J . . .
L L J M M .

Gerakan 6: I-kiri
A A B C D F
G . B C D F
G . . P P F K
G H I I I .
. H J . . .
L L J M M .

Gerakan 7: F-bawah
A A B C D .
G . B C D F
G . . P P F K
G H I I I F
. H J . . .
L L J M M .

Gerakan 8: M-kanan
A A B C D .
G . B C D F
G . . P P F K
G H I I I F
. H J . . .
L L J . M M

Gerakan 9: F-bawah
A A B C D .
G . B C D .
G . . P P F K
G H I I I F
. H J . . F
L L J . M M

Gerakan 10: H-atas
A A B C D .
G . B C D .
G H . P P F K
G H I I I F
. . J . . F
L L J . M M

Gerakan 11: M-kiri
A A B C D .
G . B C D .
G H . P P F K
G H I I I F
. . J . . F
L L J M M .

Gerakan 11: M-kiri
A A B C D .
G . B C D .
G H . P P F K
G H I I I F
. . J . . F
L L J M M .

Gerakan 12: H-atas
A A B C D .
G H B C D .
G H . P P F K
G . I I I F
. . J . . F
L L J M M .

Gerakan 13: F-bawah
A A B C D .
G H B C D .
G H . P P . K
G . I I I F
. . J . . F
L L J M M F

Gerakan 14: P-kanan
A A B C D .
G H B C D .
G H . . P P K
G . I I I F
. . J . . F
L L J M M F

Gerakan 15: P-kanan
A A B C D .
G H B C D .
G H . . . P K
G . I I I F
. . J . . F
L L J M M F

Gerakan 16: P-kanan
A A B C D .
G H B C D .
G H . . . . K
G . I I I F
. . J . . F
L L J M M F

Visited nodes: 67
Execution time: 36 ms
```

**Output dengan GBFS**

test > solusi1Astar.txt

```
Gerakan 1: D-atas
A A B . D F
. . B C D F
G P P C . F K
G H . I I I
G H J . . .
L L J M M .

Gerakan 2: I-kiri
A A B . D F
. . B C D F
G P P C . F K
G H I I I .
G H J . . .
L L J M M .

Gerakan 3: C-atas
A A B C D F
. . B C D F
G P P . . F K
G H I I I .
G H J . . .
L L J M M .

Gerakan 4: F-bawah
A A B C D .
. . B C D F
G P P . . F K
G H I I I F
G H J . . .
L L J M M .

Gerakan 5: P-kanan
A A B C D .
. . B C D F
G . P P . F K
G H I I I F
G H J . . .
L L J M M .

Gerakan 6: P-kanan
A A B C D .
. . B C D F
G . . P P F K
G H I I I F
G H J . . .
L L J M M .

Gerakan 7: F-bawah
A A B C D .
. . B C D .
G . . P P F K
G H I I I F
G H J . . F
L L J M M .

Gerakan 8: F-bawah
A A B C D .
. . B C D .
G . . P P . K
G H I I I F
G H J . . F
L L J M M F

Gerakan 9: P-kanan
A A B C D .
. . B C D .
G . . . P P K
G H I I I F
G H J . . F
L L J M M F

Gerakan 10: P-kanan
A A B C D .
. . B C D .
G . . . . P K
G H I I I F
G H J . . F
L L J M M F

Gerakan 11: P-kanan
A A B C D .
. . B C D .
G . . . . . K
G H I I I F
G H J . . F
L L J M M F

Visited nodes: 2136
Execution time: 145 ms
```

**Output dengan A***

4.2. Test Case 2

test > test2.txt

```
1  6 6
2  12
3    K
4  AABB.F
5  ...CDF
6  G.PCDF
7  GHPIII
8  GHJ...
9  LLJMM.
```

**Input, kondisi K di atas**

test > solusi2UCS.txt

```
Gerakan 1: P-atas
    K
A A B B . F
. . P C D F
G . P C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 2: B-kanan
    K
A A . B B F
. . P C D F
G . P C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 3: P-atas
    K
A A P B B F
. . P C D F
G . . C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 4: P-atas
    K
A A P B B F
. . . C D F
G . . C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 5: P-atas
    K
A A . B B F
. . . C D F
G . . C D F
G H . I I I
G H J . . .
L L J M M .

Visited nodes: 237
Execution time: 51 ms

```

**Output dengan UCS**

test > solusi2GBFS.txt

```
Gerakan 1: P-atas
    K
A A B B . F
. . P C D F
G . P C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 2: B-kanan
    K
A A . B B F
. . P C D F
G . P C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 3: P-atas
    K
A A P B B F
. . P C D F
G . . C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 4: M-kanan
    K
A A P B B F
. . P C D F
G . . C D F
G H . I I I
G H J . . .
L L J . M M

Gerakan 5: J-atas
    K
A A P B B F
. . P C D F
G . . C D F
G H J I I I
G H J . . .
L L . . M M

Gerakan 6: P-atas
    K
A A P B B F
. . . C D F
G . . C D F
G H J I I I
G H J . . .
L L . . M M

Gerakan 7: L-kanan
    K
A A P B B F
. . . C D F
G . . C D F
G H J I I I
G H J . . .
. L L . M M

Gerakan 8: M-kiri
    K
A A P B B F
. . . C D F
G . . C D F
G H J I I I
G H J . . .
. L L M M .

Gerakan 9: P-atas
    K
A A . B B F
. . . C D F
G . . C D F
G H J I I I
G H J . . .
. L L M M .

Visited nodes: 22
Execution time: 29 ms
```

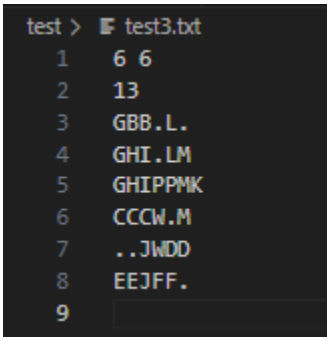
**Output dengan GBFS**

test > solusi2Astar.txt

```
Gerakan 1: B-kanan
    K
A A . B B F
. . . C D F
G . P C D F
G H P I I I
G H J . . .
L L J M M .

Gerakan 2: P-atas
    K
A A . B B F
. . P C D F
G . P C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 3: P-atas
    K
A A P B B F
. . P C D F
G . . C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 4: P-atas
    K
A A P B B F
. . . C D F
G . . C D F
G H . I I I
G H J . . .
L L J M M .

Gerakan 5: P-atas
    K
A A . B B F
. . . C D F
G . . C D F
G H . I I I
G H J . . .
L L J M M .

Visited nodes: 254
Execution time: 64 ms
```

**Output dengan A***

4.3. Test Case 3



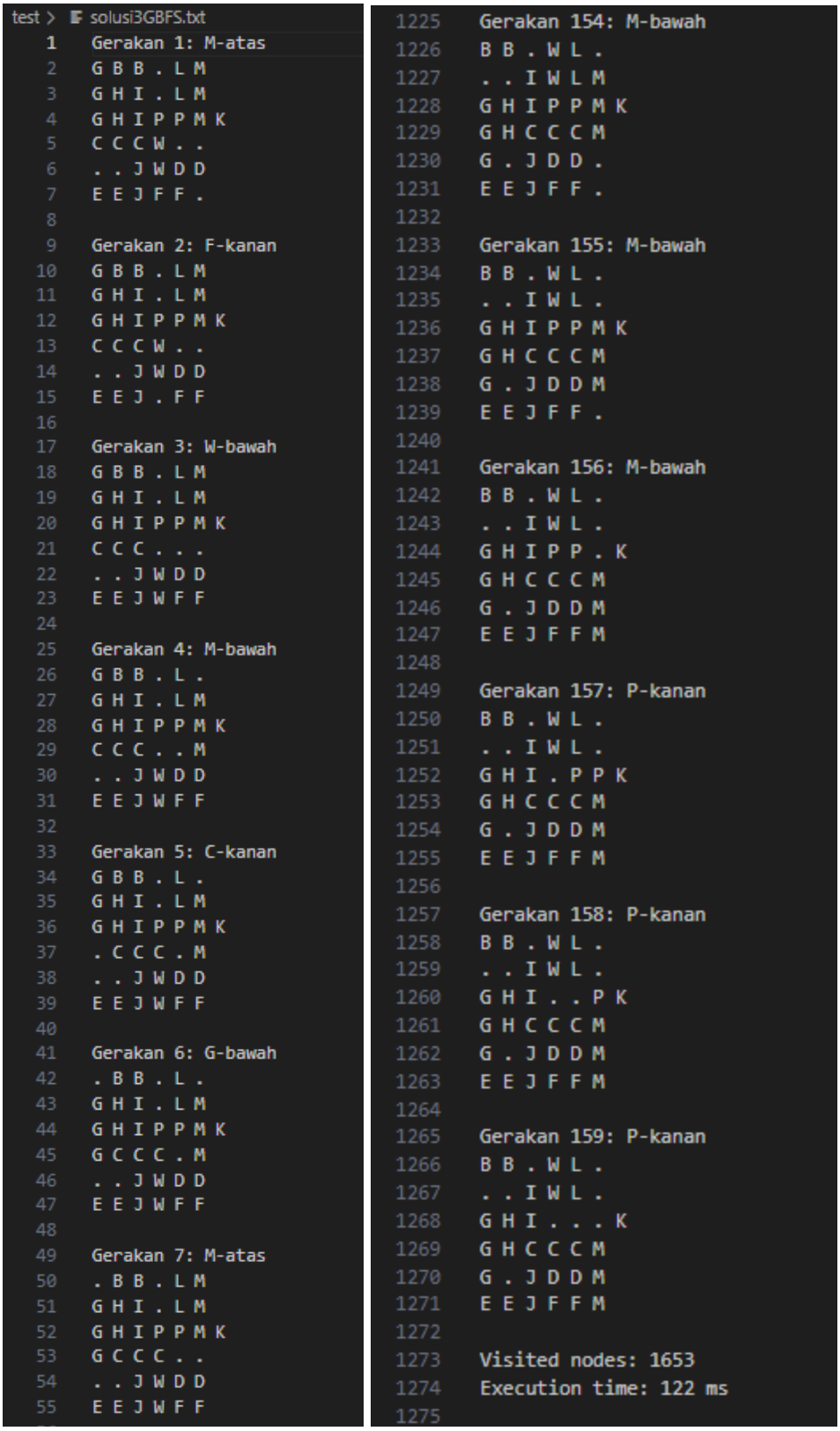

**Input, kondisi kasus kompleks, memiliki jalur solusi panjang**

test > solusi3UCS.txt

```
Gerakan 1: F-kanan
G B B . L .
G H I . L M
G H I P P M K
C C C W . M
. . J W D D
E E J . F F

Gerakan 2: W-bawah
G B B . L .
G H I . L M
G H I P P M K
C C C . . M
. . J W D D
E E J W F F

Gerakan 3: C-kanan
G B B . L .
G H I . L M
G H I P P M K
. C C C . M
. . J W D D
E E J W F F

Gerakan 4: G-bawah
. B B . L .
G H I . L M
G H I P P M K
G C C C . M
. . J W D D
E E J W F F

Gerakan 5: C-kanan
. B B . L .
G H I . L M
G H I P P M K
G . C C C M
. . J W D D
E E J W F F

Gerakan 6: B-kiri
B B . . L .
G H I . L M
G H I P P M K
G . C C C M
. . J W D D
E E J W F F

Gerakan 7: M-atas
B B . . L M
G H I . L M
G H I P P M K
G . C C C .
. . J W D D
E E J W F F
```

```
Gerakan 78: P-kanan
B B I W L .
. . I W L .
G . P P . M K
G H C C C M
G H J D D M
E E J F F .

Gerakan 79: M-bawah
B B I W L .
. . I W L .
G . P P . . K
G H C C C M
G H J D D M
E E J F F M

Gerakan 80: P-kanan
B B I W L .
. . I W L .
G . . P P . K
G H C C C M
G H J D D M
E E J F F M

Gerakan 81: P-kanan
B B I W L .
. . I W L .
G . . . P P K
G H C C C M
G H J D D M
E E J F F M

Gerakan 82: P-kanan
B B I W L .
. . I W L .
G . . . . P K
G H C C C M
G H J D D M
E E J F F M

Gerakan 83: P-kanan
B B I W L .
. . I W L .
G . . . . . K
G H C C C M
G H J D D M
E E J F F M

Visited nodes: 10176
Execution time: 220 ms
```

**Output dengan UCS, menampilkan beberapa gerakan pertama dan terakhir**



```
test > solusi3GBFS.txt
1    Gerakan 1: M-atas
2    G B B . L M
3    G H I . L M
4    G H I P P M K
5    C C C W . .
6    . . J W D D
7    E E J F F .
8
9    Gerakan 2: F-kanan
10   G B B . L M
11   G H I . L M
12   G H I P P M K
13   C C C W . .
14   . . J W D D
15   E E J . F F
16
17   Gerakan 3: W-bawah
18   G B B . L M
19   G H I . L M
20   G H I P P M K
21   C C C . . .
22   . . J W D D
23   E E J W F F
24
25   Gerakan 4: M-bawah
26   G B B . L .
27   G H I . L M
28   G H I P P M K
29   C C C . . M
30   . . J W D D
31   E E J W F F
32
33   Gerakan 5: C-kanan
34   G B B . L .
35   G H I . L M
36   G H I P P M K
37   . C C C . M
38   . . J W D D
39   E E J W F F
40
41   Gerakan 6: G-bawah
42   . B B . L .
43   G H I . L M
44   G H I P P M K
45   G C C C . M
46   . . J W D D
47   E E J W F F
48
49   Gerakan 7: M-atas
50   . B B . L M
51   G H I . L M
52   G H I P P M K
53   G C C C . .
54   . . J W D D
55   E E J W F F
```

```
1225   Gerakan 154: M-bawah
1226   B B . W L .
1227   . . I W L M
1228   G H I P P M K
1229   G H C C C M
1230   G . J D D .
1231   E E J F F .
1232
1233   Gerakan 155: M-bawah
1234   B B . W L .
1235   . . I W L .
1236   G H I P P M K
1237   G H C C C M
1238   G . J D D M
1239   E E J F F .
1240
1241   Gerakan 156: M-bawah
1242   B B . W L .
1243   . . I W L .
1244   G H I P P . K
1245   G H C C C M
1246   G . J D D M
1247   E E J F F M
1248
1249   Gerakan 157: P-kanan
1250   B B . W L .
1251   . . I W L .
1252   G H I . P P K
1253   G H C C C M
1254   G . J D D M
1255   E E J F F M
1256
1257   Gerakan 158: P-kanan
1258   B B . W L .
1259   . . I W L .
1260   G H I . . P K
1261   G H C C C M
1262   G . J D D M
1263   E E J F F M
1264
1265   Gerakan 159: P-kanan
1266   B B . W L .
1267   . . I W L .
1268   G H I . . . K
1269   G H C C C M
1270   G . J D D M
1271   E E J F F M
1272
1273   Visited nodes: 1653
1274   Execution time: 122 ms
1275
```

**Output dengan GBFS, menampilkan beberapa gerakan pertama dan terakhir**

test > solusi3Astar.txt

```
Gerakan 1: F-kanan
G B B . L .
G H I . L M
G H I P P M K
C C C W . M
. . J W D D
E E J . F F

Gerakan 2: W-bawah
G B B . L .
G H I . L M
G H I P P M K
C C C . . M
. . J W D D
E E J W F F

Gerakan 3: C-kanan
G B B . L .
G H I . L M
G H I P P M K
. C C C . M
. . J W D D
E E J W F F

Gerakan 4: G-bawah
. B B . L .
G H I . L M
G H I P P M K
G C C C . M
. . J W D D
E E J W F F

Gerakan 5: C-kanan
. B B . L .
G H I . L M
G H I P P M K
G . C C C M
. . J W D D
E E J W F F

Gerakan 6: B-kiri
B B . . L .
G H I . L M
G H I P P M K
G . C C C M
. . J W D D
E E J W F F

Gerakan 7: M-atas
B B . . L M
G H I . L M
G H I P P M K
G . C C C .
. . J W D D
E E J W F F

Gerakan 78: P-kanan
B B I W L .
. . I W L .
G . P P . M K
G H C C C M
G H J D D M
E E J F F .

Gerakan 79: M-bawah
B B I W L .
. . I W L .
G . P P . . K
G H C C C M
G H J D D M
E E J F F M

Gerakan 80: P-kanan
B B I W L .
. . I W L .
G . . P P . K
G H C C C M
G H J D D M
E E J F F M

Gerakan 81: P-kanan
B B I W L .
. . I W L .
G . . . P P K
G H C C C M
G H J D D M
E E J F F M

Gerakan 82: P-kanan
B B I W L .
. . I W L .
G . . . . P K
G H C C C M
G H J D D M
E E J F F M

Gerakan 83: P-kanan
B B I W L .
. . I W L .
G . . . . . K
G H C C C M
G H J D D M
E E J F F M

Visited nodes: 10176
Execution time: 215 ms
```

**Output dengan A*, menampilkan beberapa gerakan pertama dan terakhir**

4.4. Test Case 4

test4.txt

```
6 6
7
AAX...
..XBBB
PPC.DDK
..C...
..EEE.
......
```

**Input, kondisi tidak memiliki solusi**

solusi4UCS.txt

```
Tidak ada solusi ditemukan.
Visited nodes: 120
Execution time: 21 ms
```

**Output dengan UCS**

solusi4GBFS.txt

```
Tidak ada solusi ditemukan.
Visited nodes: 120
Execution time: 31 ms
```

**Output dengan GBFS**

solusi4Astar.txt

```
Tidak ada solusi ditemukan.
Visited nodes: 251
Execution time: 35 ms
```

**Output dengan A***

# BAB V
# ANALISIS

Berdasarkan pengujian yang sudah dilakukan terhadap beberapa kasus uji dan dengan beberapa algoritma pathfinding, algoritma A* merupakan algoritma yang cenderung memberi hasil paling optimal dibanding algoritma GBFS dan UCS. A* menjadi yang paling optimal karena mengkombinasikan keunggulan GBFS dan UCS dengan mempertimbangkan g(n) dan h(n) sehingga A* bisa memilih jalur yang tetap memprioritaskan biaya rendah dan menjanjikan untuk langkah berikut.

Pada beberapa kasus GBFS memberikan hasil yang tidak optimal dibanding algoritma lainnya namun memiliki waktu eksekusi yang cenderung lebih cepat karena cenderung mengejar biaya menuju tujuan sehingga kerap mencapai hasil yang kurang optimal.

Dari segi kompleksitas waktu teoretis, semua algoritma memiliki kompleksitas paling buruk sebesar $O(b^d)$, dengan b sebagai *branching factor* dan d kedalaman solusi. Namun, dalam praktiknya, A* seringkali lebih unggul karena mampu memfokuskan pencarian pada jalur yang paling menjanjikan, terutama jika heuristiknya *admissible* dan *consistent*. UCS unggul dalam jaminan optimalitas, dan GBFS dalam kecepatan solusi awal—meskipun tanpa jaminan mutu hasil. Pemilihan algoritma terbaik pun akhirnya bergantung pada kebutuhan spesifik seperti kecepatan, keterbatasan memori, atau keharusan mendapatkan solusi optimal.

# BAB VI
# LAMPIRAN

1. Pranala Github : https://github.com/mAlfnsy/Tucil3_13523005
2. Tabel Checklist :

| No | Poin | Ya | Tidak |
|---|---|---|---|
| 1. | Program berhasil dikompilasi tanpa kesalahan. | ✓ | |
| 2. | Program berhasil dijalankan. | ✓ | |
| 3. | Solusi yang diberikan program benar dan mematuhi aturan permainan | ✓ | |
| 4. | Program dapat membaca masukan berkas .txt dan menyimpan solusi berupa print board tahap per tahap dalam berkas .txt | ✓ | |
| 5. | [Bonus] Implementasi algoritma pathfinding alternatif | | ✓ |
| 6. | [Bonus] Implementasi 2 atau lebih heuristik alternatif | | ✓ |
| 7. | [Bonus] Program memiliki GUI | | ✓ |
| 8. | Program dan laporan dibuat (kelompok) sendiri. | ✓ | |