Tugas Kecil 3 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding

Disusun oleh:
Muhammad Iqbal Haidar (13523111)

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

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

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.

SPESIFIKASI

- 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.

- Algoritma pathfinding minimal menggunakan satu atau lebih heuristic yang ditentukan sendiri. Jika mengerjakan bonus, heuristic 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
11
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.

BAB II

ALGORITMA

Algoritma pathfinding UCS, Greedy Best First Search, dan A* pada dasarnya memiliki implementasi yang mirip namun terdapat perbedaan pada perhitungan costnya. Secara umum berikut adalah algoritma pathfinding dari program Rush Hour Solver

1. [INISIALISASI] Ambil state/kondisi awal dari puzzle dan jadikan sebagai root node dengan cost g(n) = 0 lalu set cost f(n) = cost g(n).
2. Apabila menggunakan algoritma GBFS / A* maka hitung nilai heuristik h(n) dari state tersebut dan tambahkan ke cost f(n) = g(n) + h(n) **[f(n) = Evaluation Function]**
3. Masukkan root node ke priority queue berdasarkan cost
4. Pop node dari priority queue dan catat pada visitedNode
5. Cek apakah node tersebut merupakan node tujuan, jika iya maka program selesai
6. [EKSPANSI] Iterasi setiap kemungkinan pergerakan valid dari setiap piece lalu cek apakah dengan pergerakan tersebut menghasilkan state baru yang belum ada di visitedNode
7. Apabila belum ada, maka jadikan state tersebut sebagai node baru dengan cost g(n) ditambah 1 dari cost g(n) sebelumnya dan jadikan cost f(n) = cost g(n). Apabila menggunakan GBFS maka selalu cost g(n) = 0
8. Apabila menggunakan algoritma GBFS / A* maka hitung nilai heuristik h(n) dari state tersebut dan tambahkan ke cost f(n) = g(n) + h(n) **[f(n) = Evaluation Function]**
9. Masukkan node ke priority queue berdasarkan cost
10. Ulangi langkah 4 sampai priority queue kosong, dan apabila belum ketemu node tujuan maka dikatakan puzzle tidak memiliki solusi.

Berdasarkan penjabaran Algoritma tersebut dapat dikatakan bahwa Algoritma UCS selalu menyimpan jarak dari state awal hingga state saat ini sehingga Algoritma ini dapat dipastikan selalu memberikan nilai cost optimal untuk mencapai tujuan. Namun kelebihan ini juga datang dengan trade-off yakni waktu pencarian yang lebih lama ketimbang algoritma lainnya. Algoritma ini tidak menggunakan heuristik sehingga fungsi cost total f(n) hanya bergantung pada cost g(n) yakni cost dari awal state sampai pada state saat ini. Algoritma ini memiliki perbedaan dengan algoritma BFS meskipun sama-sama menghasilkan solusi optimal, namun algoritma BFS hanya dapat menghasilkan solusi optimal ketika cost = step. Apabila cost untuk mencapai simpul tetangga tidak sama dengan step maka BFS tidak optimal, lain halnya dengan UCS yang selalu mencatat cost dari awal state sampai state saat ini, bukan mencatat banyaknya step. Urutan node yang dibangkitkan antara kedua algoritma ini juga pastinya berbeda.

Algoritma GBFS memiliki pendekatan berbeda dengan UCS, pada algoritma ini tidak mencatat cost dari state awal sampai state saat ini sehingga cost total f(n) murni hanya ditentukan oleh cost

h(n). h(n) merupakan sebuah fungsi heuristik yang menghitung kira-kira berapa nilai cost yang dibutuhkan dari state saat ini untuk mencapai node tujuan . Algoritma ini memiliki kelebihan dibandingkan UCS yakni waktu pencarian yang lebih singkat namun tentunya memiliki trade-off yakni mungkin terjebak pada local minima, irrevocable, dan sangat bergantung pada keakuratan heuristik sehingga algoritma ini tidak selalu menjamin memberikan solusi optimal dari sebuah persoalan.

Algoritma A* dapat dikatakan sebagai algoritma yang menggabungkan antara UCS dan GBFS dimana pada algoritma ini fungsi evaluasi cost f(n) = g(n) + h(n) dengan begitu algoritma ini dapat menjamin selalu memberikan solusi optimal dengan waktu pencarian yang singkat sebab heuristik h(n) mencegah algoritma mengekspan node dengan prospek masa depan yang kurang bagus. Dengan begitu algoritma ini memiliki kelebihan dari kedua algoritma sebelumnya yakni UCS dan GBFS. Pada program yang telah dibuat, penulis menggunakan dua pendekatan heuristik yakni jumlah sel antara posisi primary piece saat ini dengan exit gate dan jumlah sel yang terhalang oleh piece lain antara posisi primary piece saat ini dengan exit gate. **Heuristik pertama** admissible karena untuk mencapai goal state, jarak minimum yang harus ditempuh pastilah jarak primary piece ke exit gate. Semakin jauh jarak primary piece dari exit gate, maka pastilah primary piece semakin menjauhi pintu keluar. **Heuristik kedua** admissible karena untuk mencapai goal state, jarak minimum yang harus ditempuh sebanding dengan banyak piece yang menghalangi exit gate. Semakin banyak piece yang menghalangi exit gate, maka primary piece akan semakin sulit mencapai pintu keluar. Kedua heuristic tersebut selalu meng-*underestimate* cost sebenarnya untuk mencapai goal node $h(n) \leq h^*(n)$ sehingga heuristic admissible.

Algoritma IDS

1. Mulai dari depth 0
2. Cek apakah sudah mencapai node tujuan
3. Lakukan pencarian dengan traverse dari current node ke node anak pertama
4. Cek apakah depth sudah mencapai limit, jika iya maka backtrack ke node anak lainnya milik node parent
5. Ulangi langkah 2 sampai semua child root node terkunjungi
6. Tambahkan depth sebanyak 1 dan ulangi langkah 2
7. Jika sudah mencapai maxDepth dan belum menemukan solusi maka dikatakan solusi untuk permasalahan tersebut tidak ada

Iterative Deepening Search (IDS) adalah algoritma pencarian yang menggabungkan kelebihan dari algoritma DFS (penggunaan memori kecil) dan BFS (menemukan solusi terpendek). IDS menjalankan DFS berulang kali dengan batas kedalaman (depth limit) yang meningkat secara bertahap, dimulai dari 0 hingga mencapai solusi.

IMPLEMENTASI

Implementasi kelas Piece

```
import java.awt.Point;

public class Piece {
    private char id;
    private boolean isOrientationHorizontal;
    private boolean isPrimary;
    private int size;
    private Point head; // representasi koordinat kiri atas (x, y) = (kolom,
baris)

    public Piece(char id, boolean isOrientationHorizontal, int size, Point
head) {
        this.id = id;
        this.isOrientationHorizontal = isOrientationHorizontal;
        this.size = size;
        this.head = new Point(head); // deep copy untuk keamanan

        if (id == 'P') {
            this.isPrimary = true;
        } else {
            this.isPrimary = false;
        }
    }

    // Deep copy
    public Piece clone() {
        return new Piece(this.id, this.isOrientationHorizontal, this.size,
this.head);
    }

    public char getId() {
        return id;
    }

    public boolean isOrientationHorizontal() {
        return isOrientationHorizontal;
    }
```

```
    public boolean isPrimary() {
        return isPrimary;
    }

    public int getSize() {
        return size;
    }

    public Point getHead() {
        return new Point(head); // kembalikan salinan agar aman dari mutasi
    }

    /**
     * Menggerakkan Piece sejauh 'steps' langkah dalam arah orientasinya.
     * steps > 0 artinya ke kanan/bawah, steps < 0 ke kiri/atas
     */
    public void move(int steps) {
        if (isOrientationHorizontal) {
            head.translate(steps, 0); // geser secara horizontal
        } else {
            head.translate(0, steps); // geser secara vertikal
        }
    }
}
```

Implementasi kelas Board

```
import java.util.*;
import java.awt.Point;

public class Board {
    private char[][] buf;
    private int sizeX, sizeY;
    private Point exitGate;
    private List<Piece> pieces;
    private int kPos; // 1 Kiri, 2 Atas, 3 Kanan, 4 Bawah

    public Board(int sizeX, int sizeY, Point exitGate, List<Piece> pieces, int
kPos) {
        this.sizeX = sizeX;
        this.sizeY = sizeY;
```

```
        this.exitGate = new Point(exitGate);
        this.kPos = kPos;
        this.pieces = new ArrayList<>();
        for (Piece p : pieces) {
            this.pieces.add(p.clone()); // Deep copy
        }
        rebuildBuffer();
    }

    // Menyalin papan dan semua komponennya (deep copy)
    public Board clone() {
        return new Board(this.sizeX, this.sizeY, this.exitGate, this.pieces,
this.kPos);
    }

    // Mengisi map (buf) dengan piece
    public void rebuildBuffer() {
        buf = new char[sizeY][sizeX];
        for (int y = 0; y < sizeY; y++) {
            Arrays.fill(buf[y], '.');
        }

        for (Piece p : pieces) {
            Point h = p.getHead();
            for (int i = 0; i < p.getSize(); i++) {
                int x = h.x + (p.isOrientationHorizontal() ? i : 0);
                int y = h.y + (p.isOrientationHorizontal() ? 0 : i);
                // System.out.println("y: " + y + " x: " + x + " piece id: " +
p.getId());
                buf[y][x] = p.getId();
            }
        }
    }

    public Piece findPiece(char pieceId) { // Ini aman harusnya karena
terbatas max 24 element
        for (Piece p : pieces) {
            if (p.getId() == pieceId) {
                return p;
            }
        }
```

```
        System.out.println("Piece tidak ditemukan [findPiece()]");
        return null;
    }

    public void movePiece(char pieceId, int step) {
        Piece target = findPiece(pieceId);

        if (target != null) {
            target.move(step);
            rebuildBuffer();
        }
    }

    public List<Integer> calcPossibleMove(char pieceId) {
        List<Integer> possibleMoves = new ArrayList<>();
        Piece target = findPiece(pieceId);

        // Cek ke arah negatif
        for (int i = 1;; i++) {
            if (canMove(target, -i)) {
                possibleMoves.add(-i);
            } else {
                break;
            }
        }

        // Cek ke arah positif
        for (int i = 1;; i++) {
            if (canMove(target, i)) {
                possibleMoves.add(i);
            } else {
                break;
            }
        }

        return possibleMoves;
    }

    private boolean canMove(Piece piece, int step) {
        Point h = piece.getHead();
        int x = h.x, y = h.y;
```

```
        int size = piece.getSize();

        for (int i = 0; i < size; i++) {
            int checkX = x + (piece.isOrientationHorizontal() ? i : 0);
            int checkY = y + (piece.isOrientationHorizontal() ? 0 : i);

            int moveX = checkX + (piece.isOrientationHorizontal() ? step : 0);
            int moveY = checkY + (piece.isOrientationHorizontal() ? 0 : step);

            if (moveX < 0 || moveX >= sizeX || moveY < 0 || moveY >= sizeY) {
                return false;
            }

            if (buf[moveY][moveX] != '.' && buf[moveY][moveX] !=
piece.getId()) {
                return false;
            }
        }

        return true;
    }

    public boolean isWin() {
        for (Piece p : pieces) {
            if (p.isPrimary()) {
                Point tail = p.getHead();
                int size = p.getSize();

                if (p.isOrientationHorizontal()) {
                    tail.translate(size - 1, 0);
                } else {
                    tail.translate(0, size - 1);
                }

                return p.getHead().equals(exitGate) || tail.equals(exitGate);
            }
        }
        return false;
    }

    public List<Piece> getPieces() { // Hanya untuk cari id doang kan
```

```
        return this.pieces;
    }

    public int getKPos() {
        return this.kPos;
    }

    public boolean equals(Board b) { // Ini harus di optimisasi sih gila
        // System.out.println("sizeX: " + sizeX + ", sizeY: " + sizeY);
        for (int i = 0; i < sizeY; i++) {
            for (int j = 0; j < sizeX; j++) {
                // System.out.println("(i: " + i + ", j: " + j + ")");
                if (b.buf[i][j] != this.buf[i][j]) {
                    return false;
                }
            }
        }
        return true;
    }

    public String toString2(char id) {
        final String RESET = "\u001B[0m";
        final String RED = "\u001B[48;5;196m"; // Primary
        final String CYAN = "\u001B[48;5;226m"; // Moveable
        final String GREEN = "\u001B[48;5;46m"; // Exit

        StringBuilder sb = new StringBuilder();
        if (kPos == 1) { // Kiri
            for (int i = 0; i < sizeY; i++) {
                if (i == exitGate.y) {
                    sb.append(GREEN).append('K').append(RESET);
                } else {
                    sb.append(' ');
                }

                for (int j = 0; j < sizeX; j++) {
                    char curChar = buf[i][j];
                    if (curChar == id && curChar != 'P') {
                        sb.append(CYAN).append(curChar).append(RESET);
                    } else if (curChar == 'P') {
                        sb.append(RED).append(curChar).append(RESET);
```

```
                } else {
                    sb.append(curChar);
                }
            }
            sb.append('\n');
        }
    } else if (kPos == 2) { // Atas
        for (int i = 0; i < sizeX; i++) {
            if (i == exitGate.x) {
                sb.append(GREEN).append('K').append(RESET);
            } else {
                sb.append(' ');
            }
        }
        sb.append('\n');

        for (int i = 0; i < sizeY; i++) {
            for (int j = 0; j < sizeX; j++) {
                char curChar = buf[i][j];
                if (curChar == id && curChar != 'P') {
                    sb.append(CYAN).append(curChar).append(RESET);
                } else if (curChar == 'P') {
                    sb.append(RED).append(curChar).append(RESET);
                } else {
                    sb.append(curChar);
                }
            }
            sb.append('\n');
        }

    } else if (kPos == 3) { // Kanan
        for (int i = 0; i < sizeY; i++) {
            for (int j = 0; j < sizeX; j++) {
                char curChar = buf[i][j];
                if (curChar == id && curChar != 'P') {
                    sb.append(CYAN).append(curChar).append(RESET);
                } else if (curChar == 'P') {
                    sb.append(RED).append(curChar).append(RESET);
                } else {
                    sb.append(curChar);
                }
```

```
                }

                if (i == exitGate.y) {
                    sb.append(GREEN).append('K').append(RESET);
                }
                sb.append('\n');
            }

        } else if (kPos == 4) { // Bawah
            for (int i = 0; i < sizeY; i++) {
                for (int j = 0; j < sizeX; j++) {
                    char curChar = buf[i][j];
                    if (curChar == id && curChar != 'P') {
                        sb.append(CYAN).append(curChar).append(RESET);
                    } else if (curChar == 'P') {
                        sb.append(RED).append(curChar).append(RESET);
                    } else {
                        sb.append(curChar);
                    }
                }
                sb.append('\n');
            }

            for (int i = 0; i < sizeX; i++) {
                if (i == exitGate.x) {
                    sb.append(GREEN).append('K').append(RESET);
                } else {
                    sb.append(' ');
                }
            }
            sb.append('\n');
        }

        return sb.toString();
    }

    public String toString3(char id) {
        StringBuilder sb = new StringBuilder();
        if (kPos == 1) { // Kiri
            for (int i = 0; i < sizeY; i++) {
                if (i == exitGate.y) {
```

```
                    sb.append('K');
                } else {
                    sb.append(' ');
                }

                for (int j = 0; j < sizeX; j++) {
                    char curChar = buf[i][j];
                    if (curChar == id && curChar != 'P') {
                        sb.append(curChar);
                    } else if (curChar == 'P') {
                        sb.append(curChar);
                    } else {
                        sb.append(curChar);
                    }
                }
                sb.append('\n');
            }
        } else if (kPos == 2) { // Atas
            for (int i = 0; i < sizeX; i++) {
                if (i == exitGate.x) {
                    sb.append('K');
                } else {
                    sb.append(' ');
                }
            }
            sb.append('\n');

            for (int i = 0; i < sizeY; i++) {
                for (int j = 0; j < sizeX; j++) {
                    char curChar = buf[i][j];
                    if (curChar == id && curChar != 'P') {
                        sb.append(curChar);
                    } else if (curChar == 'P') {
                        sb.append(curChar);
                    } else {
                        sb.append(curChar);
                    }
                }
                sb.append('\n');
            }
```

```
        } else if (kPos == 3) { // Kanan
            for (int i = 0; i < sizeY; i++) {
                for (int j = 0; j < sizeX; j++) {
                    char curChar = buf[i][j];
                    if (curChar == id && curChar != 'P') {
                        sb.append(curChar);
                    } else if (curChar == 'P') {
                        sb.append(curChar);
                    } else {
                        sb.append(curChar);
                    }
                }

                if (i == exitGate.y) {
                    sb.append('K');
                }
                sb.append('\n');
            }

        } else if (kPos == 4) { // Bawah
            for (int i = 0; i < sizeY; i++) {
                for (int j = 0; j < sizeX; j++) {
                    char curChar = buf[i][j];
                    if (curChar == id && curChar != 'P') {
                        sb.append(curChar);
                    } else if (curChar == 'P') {
                        sb.append(curChar);
                    } else {
                        sb.append(curChar);
                    }
                }
                sb.append('\n');
            }

            for (int i = 0; i < sizeX; i++) {
                if (i == exitGate.x) {
                    sb.append('K');
                } else {
                    sb.append(' ');
                }
            }
```

```
            sb.append('\n');
        }

        return sb.toString();
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (char[] row : buf) {
            for (char c : row) {
                sb.append(c);
            }
            sb.append('\n');
        }
        return sb.toString();
    }

    public int calcH() {
        Piece prim = null;
        for (Piece p : pieces) {
            if (p.isPrimary()) {
                prim = p;
            }
        }

        int block = 0;
        int exitX = exitGate.x;
        int exitY = exitGate.y;
        int primX = prim.getHead().x;
        int primY = prim.getHead().y;
        char primId = prim.getId();

        if (prim.isOrientationHorizontal()) {
            int dx = exitX - primX;
            if (dx > 0) {
                for (int i = 0; i < dx; i++) {
                    if (buf[primY][primX + i + 1] != '.' && buf[primY][primX +
i + 1] != primId) {
                        block++;
                    }
                }
```

```
            } else {
                dx *= -1;
                for (int i = 0; i < dx; i++) {
                    if (buf[primY][primX - i - 1] != '.' && buf[primY][primX -
i - 1] != primId) {
                        block++;
                    }
                }
            }
        } else {
            int dy = exitY - primY;
            if (dy > 0) {
                for (int i = 0; i < dy; i++) {
                    if (buf[primY + i + 1][primX] != '.' && buf[primY + i +
1][primX] != primId) {
                        block++;
                    }
                }
            } else {
                dy *= -1;
                for (int i = 0; i < dy; i++) {
                    if (buf[primY - i - 1][primX] != '.' && buf[primY - i -
1][primX] != primId) {
                        block++;
                    }
                }
            }
        }

        return block;
    }

    public int calcH2() {
        Piece prim = null;
        for (Piece p : pieces) {
            if (p.isPrimary()) {
                prim = p;
            }
        }

        int exitX = exitGate.x;
```

```
        int exitY = exitGate.y;
        int primX = prim.getHead().x;
        int primY = prim.getHead().y;
        int primSize = prim.getSize();

        if (prim.isOrientationHorizontal()) {
            int dxHead = Math.abs(exitX - primX);
            int dxTail = Math.abs(exitX - (primX + primSize - 1));
            if (dxHead < dxTail) {
                return dxHead;
            } else {
                return dxTail;
            }
        } else {
            int dyHead = Math.abs(exitY - primY);
            int dyTail = Math.abs(exitY - (primY + primSize - 1));
            if (dyHead < dyTail) {
                return dyHead;
            } else {
                return dyTail;
            }
        }
    }

    public String toHashString() {
        StringBuilder sb = new StringBuilder();
        for (char[] row : buf) {
            sb.append(row);
        }
        return sb.toString();
    }

}
```

Implementasi kelas Node

```
import java.util.*;

public class Node {
    private int costGn; // cost g(n)
    private int cost; // total cost f(n)
```

```
    private String what; // deskripsi langkah terakhir sebelum sampai ke state
ini
    private Board state; // keadaan papan saat ini
    private ArrayList<Integer> path; // jejak langkah berupa ID node
sebelumnya

    public Node(Board state) { // By default membuat root node
        this.costGn = 0;
        this.cost = 0;
        this.what = "Papan awal";
        this.state = state.clone();
        this.path = new ArrayList<>();
    }

    public Node clone() { // Deep copy
        Node copy = new Node(this.state.clone());
        copy.costGn = this.costGn;
        copy.cost = this.cost;
        copy.what = this.what;
        copy.path = new ArrayList<>(this.path);
        return copy;
    }

    public int getCostGn() {
        return costGn;
    }

    public void setCostGn(int c) {
        this.costGn = c;
    }

    public int getCost() {
        return cost;
    }

    public void setCost(int c) {
        this.cost = c;
    }

    public String getWhat() {
        return what;
```

```
    }

    public void setWhat(String w) {
        this.what = w;
    }

    public Board getState() {
        return state;
    }

    public void setState(Board b) {
        this.state = b;
    }

    public ArrayList<Integer> getPath() {
        return path;
    }

    public boolean isWinning() {
        return state.isWin();
    }
}
```

Implementasi algoritma Solver

```
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Set;

public class Solver {
    private PriorityQueue<Node> pq;
    private ArrayList<Node> visitedNodes;
    Set<String> visitedStates;

    public Solver(Node rootNode) { // Selalu minta root nodenya sebagai start
```

```
        this.pq = new PriorityQueue<>(Comparator.comparingInt(Node::getCost));
        pq.add(rootNode);
        this.visitedNodes = new ArrayList<>();
        this.visitedStates = new HashSet<>();
    }

    public boolean startIDDFS(Node root, int maxDepth, int[] visitedNodeCtr) {
        for (int depth = 0; depth <= maxDepth; depth++) {
            visitedStates.clear();
            visitedNodes.clear();

            visitedNodes.add(root);
            visitedStates.add(root.getState().toHashString());
            visitedNodeCtr[0]++;

            if (depthLimitedSearch(root, depth, 0, visitedNodeCtr)) {
                return true;
            }
        }

        return false;
    }

    private boolean depthLimitedSearch(Node node, int limit, int nodeIndex,
int[] visitedNodeCtr) {
        visitedNodeCtr[0]++;
        if (node.isWinning()) {
            return true;
        }

        if (limit == 0) return false;

        for (Piece p : node.getState().getPieces()) {
            for (int move : node.getState().calcPossibleMove(p.getId())) {
                char pieceId = p.getId();
                Board newBoard = node.getState().clone();
                newBoard.movePiece(pieceId, move);

                String hash = newBoard.toHashString();
                if (visitedStates.contains(hash)) continue;
```

```
                Node child = node.clone();
                child.setState(newBoard);
                child.setWhat(pieceId + "-" + (p.isOrientationHorizontal() ?
                    (move > 0 ? "kanan" : "kiri") :
                    (move > 0 ? "bawah" : "atas")));

                child.getPath().add(nodeIndex);
                visitedNodes.add(child);
                int childIndex = visitedNodes.size() - 1;
                visitedStates.add(hash);

                if (depthLimitedSearch(child, limit - 1, childIndex,
visitedNodeCtr)) {
                    return true;
                }

                visitedStates.remove(hash);
                visitedNodes.remove(visitedNodes.size() - 1);
            }
        }

        return false;
    }

    public void printSolution() {
        if (visitedNodes.isEmpty() || !visitedNodes.get(visitedNodes.size() -
1).isWinning()) {
            System.out.println("Tidak ada solusi.");
            return;
        }

        Node winningNode = visitedNodes.get(visitedNodes.size() - 1);
        List<Integer> solutionPath = new ArrayList<>();

        // Rekonstruksi jalur dari node pemenang ke root
        int currentIndex = visitedNodes.size() - 1;
        while (!winningNode.getPath().isEmpty()) {
            solutionPath.add(0, currentIndex); // Tambahkan di awal untuk
urutan langkah yang benar
            currentIndex =
winningNode.getPath().get(winningNode.getPath().size() - 1);
```

```
            winningNode = visitedNodes.get(currentIndex);
        }
        solutionPath.add(0, 0); // Tambahkan root node

        // Cetak langkah-langkah solusi
        System.out.println("Solusi ditemukan dalam " + (solutionPath.size() -
1) + " langkah:");
        for (int i = 1; i < solutionPath.size(); i++) {
            Node node = visitedNodes.get(solutionPath.get(i));
            System.out.println("Langkah " + i + ": " + node.getWhat());
            // Anda juga bisa mencetak representasi board jika diinginkan
            // System.out.println(node.getState().toString());
        }
    }

    /*
     * 1 = UCS
     * 2 = GBFS
     * 3 = A*
     * jika return true maka ketemu, jika false maka tidak ketemu
     */
    public boolean start(int algo, int heu, int[] visitedNodeCtr) {
        // int ctrV = 0;
        // int ctrN = 0;
        // System.out.println(pq.size());
        while (!pq.isEmpty()) {
            // System.out.print("Masuk woy");
            Node currentNode = pq.poll();
            // System.out.println("Node in pq: " + pq.size());

            int currentId = visitedNodes.size();
            visitedNodes.add(currentNode);
            // ctrV++;
            // System.out.println("Visited node: " + ctrV);

            if (currentNode.isWinning()) {
                visitedNodeCtr[0] = visitedNodes.size();
                return true;
            }

            for (Piece p : currentNode.getState().getPieces()) {
```

```
                for (int move :
currentNode.getState().calcPossibleMove(p.getId())) {
                    char pieceId = p.getId();
                    Board candidateBoard = currentNode.getState().clone();
                    boolean isNotVisited = true;

                    candidateBoard.movePiece(pieceId, move);
                    String candidateHash = candidateBoard.toHashString();
                    if (!visitedStates.contains(candidateHash)) {
                        visitedStates.add(candidateHash);
                    } else {
                        isNotVisited = false;
                    }
                    // ctrN++;
                    // System.out.println("Checked node: " + ctrN);
                    // System.out.println("Queue size: " + pq.size());

                    if (isNotVisited) {
                        // System.out.println(candidateBoard.toString());
                        Node candidateNode = currentNode.clone();
                        candidateNode.getPath().add(currentId);
                        candidateNode.setState(candidateBoard);

                        String arah;
                        if (p.isOrientationHorizontal()) {
                            arah = (move > 0) ? "kanan" : "kiri";
                        } else {
                            arah = (move > 0) ? "bawah" : "atas";
                        }
                        String what = pieceId + "-" + arah;
                        candidateNode.setWhat(what);

                        switch (algo) {
                            case 1: // UCS
                                int currentCostGn = candidateNode.getCostGn()
+ 1;

                                candidateNode.setCostGn(currentCostGn);
                                candidateNode.setCost(currentCostGn);
                                break;
                            case 2: // GBFS
                                int heuValue;
```

```
                                if (heu == 1) {
                                    heuValue =
candidateNode.getState().calcH();
                                } else {
                                    heuValue =
candidateNode.getState().calcH2();
                                }

                                candidateNode.setCostGn(-1);
                                candidateNode.setCost(heuValue);
                                break;
                            case 3: // A*
                                int heuValue2;
                                if (heu == 1) {
                                    heuValue2 =
candidateNode.getState().calcH();
                                } else {
                                    heuValue2 =
candidateNode.getState().calcH2();
                                }

                                int currentCostGn2 = candidateNode.getCostGn()
+ 1;
                                candidateNode.setCostGn(currentCostGn2);
                                candidateNode.setCost(currentCostGn2 +
heuValue2);
                                break;

                            default:
                                break;
                        }

                        pq.add(candidateNode);
                    }
                }
            }
        }
        visitedNodeCtr[0] = visitedNodes.size();
        return false;
    }
```

```
    public void display() {
        int finishNodeIdx = visitedNodes.size() - 1;
        Node finishNode = visitedNodes.get(finishNodeIdx);

        int ctr = 0;
        for (int i : finishNode.getPath()) {
            Node iter = visitedNodes.get(i);
            char id = iter.getWhat().charAt(0);
            if (ctr == 0) {
                System.out.println("Papan Awal");
            } else {
                System.out.println("Gerakan " + ctr + ": " + iter.getWhat());
            }
            System.out.println(iter.getState().toString2(id));
            ctr++;
        }

        char id = finishNode.getWhat().charAt(0);
        System.out.println("Gerakan " + ctr + ": " + finishNode.getWhat());
        System.out.println(finishNode.getState().toString2(id));
    }

    public void display2(PrintWriter pw) {
        int finishNodeIdx = visitedNodes.size() - 1;
        Node finishNode = visitedNodes.get(finishNodeIdx);

        int ctr = 0;
        for (int i : finishNode.getPath()) {
            Node iter = visitedNodes.get(i);
            char id = iter.getWhat().charAt(0);
            if (ctr == 0) {
                pw.println("Papan Awal");
            } else {
                pw.println("Gerakan " + ctr + ": " + iter.getWhat());
            }
            pw.println(iter.getState().toString3(id));
            ctr++;
        }

        char id = finishNode.getWhat().charAt(0);
        pw.println("Gerakan " + ctr + ": " + finishNode.getWhat());
```

```
        pw.println(finishNode.getState().toString3(id));
    }
}
```

Implementasi kelas Parser

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.awt.Point;

public class Parser {
    private static Point findK(File file) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(file));
        String line;
        int y = 0;

        while ((line = br.readLine()) != null) {
            for (int x = 0; x < line.length(); x++) {
                if (line.charAt(x) == 'K') {
                    br.close();
                    return new Point(x, y - 2);
                }
            }
            y++;
        }

        br.close();
        return null;
    }

    public static Board parseBoard(String path) throws IOException {
        File file = new File(path);
        List<String> lines = new ArrayList<>();
        BufferedReader br = new BufferedReader(new FileReader(file));
        String line;
```

```
        while ((line = br.readLine()) != null) {
            lines.add(line);
        }
        br.close();

        Point kPos = findK(file);
        if (kPos == null) {
            throw new IllegalArgumentException("Tidak ditemukan karakter 'K'
di file");
        }

        String[] sizeParts = lines.get(0).split(" ");
        int sizeX = Integer.parseInt(sizeParts[0]);
        int sizeY = Integer.parseInt(sizeParts[1]);

        int kSisi;
        char[][] buf = new char[sizeY][sizeX];
        Point exit = null;

        if (kPos.x == 0 && lines.size() == sizeY + 2) { // K di kiri
            exit = new Point(0, kPos.y);
            kSisi = 1;
            // baca buf dari kolom 1 sampai sizeX
            for (int y = 0; y < sizeY; y++) {
                String row = lines.get(y + 2); // asumsi papan mulai baris
ke-2
                for (int x = 0; x < sizeX; x++) {
                    buf[y][x] = row.charAt(x + 1); // offset kolom +1 karena
kol 0 K/spasi
                }
            }
        } else if (kPos.y == 0 && lines.size() == sizeY + 3) { // K di atas
            exit = new Point(kPos.x, 0);
            kSisi = 2;
            // baca buf dari baris 1 sampai sizeY
            for (int y = 0; y < sizeY; y++) {
                String row = lines.get(y + 3); // asumsi papan mulai baris
ke-3 (karena K di baris 0)
                for (int x = 0; x < sizeX; x++) {
                    buf[y][x] = row.charAt(x);
                }
```

```
            }
        } else if (kPos.x == sizeX && lines.size() == sizeY + 2) { // K di
kanan (asumsi K di kolom sizeX + 1)
            exit = new Point(sizeX - 1, kPos.y);
            kSisi = 3;
            // baca buf biasa
            for (int y = 0; y < sizeY; y++) {
                String row = lines.get(y + 2);
                for (int x = 0; x < sizeX; x++) {
                    buf[y][x] = row.charAt(x);
                }
            }
        } else if (kPos.y == sizeY && lines.size() == sizeY + 3) { // K di
bawah (asumsi K di baris sizeY + 2)
            exit = new Point(kPos.x, sizeY - 1);
            kSisi = 4;
            // baca buf biasa
            for (int y = 0; y < sizeY; y++) {
                String row = lines.get(y + 2);
                for (int x = 0; x < sizeX; x++) {
                    buf[y][x] = row.charAt(x);
                }
            }
        } else {
            throw new IllegalArgumentException("Posisi 'K' tidak sesuai
asumsi");
        }

        // Buat pieces dari buf
        Map<Character, Piece> pieceMap = new HashMap<>();
        boolean[][] visited = new boolean[sizeY][sizeX];

        for (int y = 0; y < sizeY; y++) {
            for (int x = 0; x < sizeX; x++) {
                char c = buf[y][x];
                if (c != '.' && !visited[y][x]) {
                    visited[y][x] = true;

                    // Cek horizontal
                    int len = 1;
                    int nx = x + 1;
```

```
                    while (nx < sizeX && buf[y][nx] == c) {
                        visited[y][nx] = true;
                        len++;
                        nx++;
                    }
                    if (len > 1) {
                        pieceMap.put(c, new Piece(c, true, len, new Point(x,
y)));
                        continue;
                    }

                    // Cek vertical
                    len = 1;
                    int ny = y + 1;
                    while (ny < sizeY && buf[ny][x] == c) {
                        visited[ny][x] = true;
                        len++;
                        ny++;
                    }
                    if (len > 1) {
                        pieceMap.put(c, new Piece(c, false, len, new Point(x,
y)));
                    }
                }
            }
        }

        return new Board(sizeX, sizeY, exit, new
ArrayList<>(pieceMap.values()), kSisi);
    }
}
```

BAB III

TESTING

Uniform Cost Search (UCS)			
No	File	Input	Output
1.	test9.txt	6 6 12 ABBCCC A.D.EE PPD..FK .LIIHF .LJJHG MMM.HG	Gerakan 29: F-atas BBCCCF EED..F ..DPP.K .LIIHG ALJJHG AMMMH. Gerakan 30: P-kanan BBCCCF EED..F ..D.PPK .LIIHG ALJJHG AMMMH. Jumlah node dikunjungi: 3458 Waktu eksekusi: 82ms

2.	test10.txt	6 6
		9
		  K
		...IBB
		AAAI..
		H..DGG
		H.EDCC
		..E..P
		.FF..P
			Gerakan 5: B-kiri
			  K
			BB....
			AAAI..
			HGGI..
			H.ECC.
			..ED.P
			.FFD.P
			
			Gerakan 6: P-atas
			  K
			BB...P
			AAAI.P
			HGGI..
			H.ECC.
			..ED..
			.FFD..
			
			Jumlah node dikunjungi: 1299
			Waktu eksekusi: 50ms

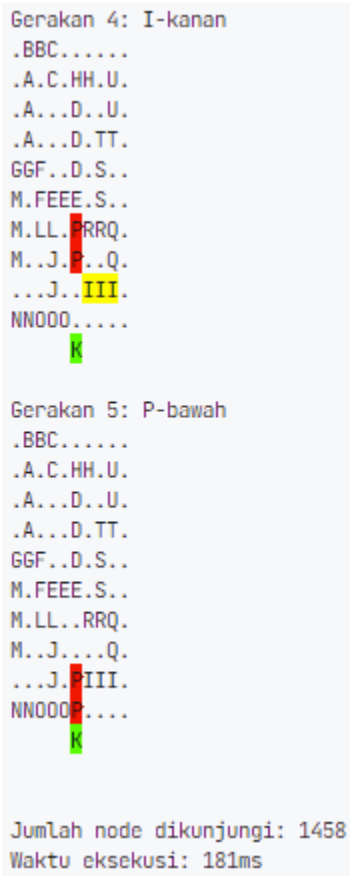
3.	test11.txt	6 6 9 B..C.. BAACI. ..HHI. G.F.DD G.F..E KG.FPPE	Gerakan 4: F-atas B..CI. BAACI. G.FHH. G.F.DD G.F..E K...PPE Gerakan 5: P-kiri B..CI. BAACI. G.FHH. G.F.DD G.F..E KPP...E Jumlah node dikunjungi: 290 Waktu eksekusi: 23ms
4.	test12.txt	6 6 10 ....BP AAA..P H..DGG H.EDCC R.E... RFF.QQ K	Gerakan 4: G-kiri ...D.P AAAD.P H..GG. H.ECC. R.E... RFFQQ. K Gerakan 5: P-bawah ...D.. AAAD.. H..GG. H.ECC. R.E..P RFFQQP K Jumlah node dikunjungi: 551 Waktu eksekusi: 30ms

Greedy Best First Search (GBFS)			
No	File	Input	Output
1.	test5.txt	6 6 10 AAA.JL BBB.JL CPP.JIK C..F.I C.EFGG DDE.HH	Gerakan 52: L-atas CAAA.L C.BBBL C.EPP.K ..EFJ. GG.FJI DDHHJI Gerakan 53: P-kanan CAAA.L C.BBBL C.E.PPK ..EFJ. GG.FJI DDHHJI Jumlah node dikunjungi: 1506 Waktu eksekusi: 55ms

2.	test6.txt	6 6
		8
		  K
		....BB
		AAA...
		H..DGG
		H.EDCC
		..E.P.
		.FF.P.
			Gerakan 7: C-kiri
			  K
			.BB...
			AAA...
			H.GG..
			H.CC..
			..EDP.
			FFEDP.
			
			Gerakan 8: P-atas
			  K
			.BB.P.
			AAA.P.
			H.GG..
			H.CC..
			..ED..
			FFED..
			
			Jumlah node dikunjungi: 38
			Waktu eksekusi: 15ms

3.	test7.txt	6 6 8 KB..CPP BA.C.. .A.C.. G..DDD G....E FFFHHE	Gerakan 16: B-bawah K.PP... BA.... BA.C.E DDDC.E G..C.. GFFFHH Gerakan 17: P-kiri KPP.... BA.... BA.C.E DDDC.E G..C.. GFFFHH Jumlah node dikunjungi: 32 Waktu eksekusi: 15ms
4.	test8.txt	10 10 23 .BBC..... PA.C.HH.U. PA...D..U. .A...D.TT. GGF..D.S.. ..FEEE.S.. ..LL.V..RR MMMJ.V.... ...J.IIIQ. NNNJ.OOOQ. K	Gerakan 223: M-kanan .A...BB... .AF..HH... .AFC.D.... ...C.D.TT. GG.J.D..U. P..JEEE.U. PLLJ.V.SRR .MMM.V.SQ. .....IIIQ. .NNN...OOO K Gerakan 224: P-bawah .A...BB... .AF..HH... .AFC.D.... ...C.D.TT. GG.J.D..U. ...JEEE.U. .LLJ.V.SRR .MMM.V.SQ. P....IIIQ. PNNN...OOO K Jumlah node dikunjungi: 11341 Waktu eksekusi: 1826ms

A*			
No	File	Input	Output
1.	test1.txt	6 6 12 ABB.E. ACD.EF ACDPPFK LLLI.F ..JIGG MMJHH.	Gerakan 50: F-bawah BBDIE. ..DIE. A..PP.K ACLLLF ACJGGF MMJHHF Gerakan 51: P-kanan BBDIE. ..DIE. A...PPK ACLLLF ACJGGF MMJHHF Jumlah node dikunjungi: 2830 Waktu eksekusi: 70ms
2.	test2.txt	6 6 7 K .....B AAAAGB P..DG. P.EDCC ..E... .FF...	Gerakan 6: A-kanan K .....B .AAAAB P..... PCC.G. ..EDG. ..EDFF Gerakan 7: P-atas K P....B PAAAAB ...... .CC.G. ..EDG. ..EDFF Jumlah node dikunjungi: 415 Waktu eksekusi: 25ms

3.	test3.txt	6 6 7 .BBC.. .A.C.. K.APP.D .A...D GGF..D ..FEEE	
4.	test4.txt	10 10 21 .BBC..... .A.C.HH.U. .A...D..U. .A...D.TT. GGF..D.S.. M.FEEE.S.. M.LL.P..RR M..J.P.... ...J.IIIQ. NN...OOOQ. K	

Iterative Deepening Search (IDS)			
No	File	Input	Output
1.	test13.txt	6 6 10 .A..DC .A..DC PPLJ..K F.LJI. FGG.I. FHHH..	Gerakan 2: L-atas .ALJDC .ALJDC PP....K F...I. FGG.I. FHHH.. Gerakan 3: P-kanan .ALJDC .ALJDC ....PPK F...I. FGG.I. FHHH.. Jumlah node dikunjungi: 2517 Waktu eksekusi: 28ms Hasil disimpan di ../data/result.txt
2.	test14.txt	6 6 10 K ...IBB AAAI.G HH.D.G .PEDCC .PE... .FF...	Gerakan 7: A-kanan K ....BB ..AAA. ...IHH .PEICC .PED.G .FFD.G Gerakan 8: P-atas K .P..BB .PAAA. ...IHH ..EICC ..ED.G .FFD.G Jumlah node dikunjungi: 9778666 Waktu eksekusi: 14052ms Hasil disimpan di ../data/result.txt

3.	test15.txt	6 6 9 B.IIJ. B.C.J. AAC..D KG.CPPD G....E FFFHHE	Gerakan 8: C-atas B.CII. B.C.JD AAC.JD K...PPE G....E GFFFHH Gerakan 9: P-kiri B.CII. B.C.JD AAC.JD KPP...E G....E GFFFHH Jumlah node dikunjungi: 3799767 Waktu eksekusi: 6015ms Hasil disimpan di ../data/result.txt
4.	test16.txt	6 6 8 ..P.B. AAP... H..DGG H..DCC R.EE.C RFF.QQ K	Gerakan 6: F-kiri ..... HAA... H.PDGG R.PDCC R..EE. FF..QQ K Gerakan 7: P-bawah ..... HAA... H..DGG R..DCC R.PEE. FFP.QQ K Jumlah node dikunjungi: 2978338 Waktu eksekusi: 4056ms Hasil disimpan di ../data/result.txt

BAB IV

ANALISIS

Percobaan telah dilakukan untuk menganalisis performa tiga algoritma pathfinding yang diimplementasikan, yaitu Uniform Cost Search (UCS), Greedy Best First Search (GBFS), dan A*. Ketiga algoritma diuji pada berbagai konfigurasi papan puzzle Rush Hour dengan tingkat kesulitan yang bervariasi. Hasil pengujian menunjukkan bahwa UCS memiliki waktu eksekusi yang cenderung lebih lama dibandingkan GBFS dan A*, terutama pada puzzle dengan jumlah simpul solusi yang lebih dalam. Hal ini terjadi karena UCS menjelajahi seluruh node berdasarkan biaya path sejauh ini (g(n)), tanpa mempertimbangkan arah tujuan. Di sisi lain, GBFS lebih cepat karena menggunakan fungsi heuristik untuk mengarahkan pencarian ke arah goal, namun tidak menjamin solusi optimal. A* berhasil memberikan performa terbaik secara keseluruhan dengan solusi yang lebih cepat dan tetap optimal karena menggabungkan g(n) dan heuristik h(n).

Dari sisi kompleksitas, UCS dan A* keduanya memiliki kompleksitas waktu $O(b^m)$ dalam kasus terburuk, dengan b adalah maksimum percabangan dari suatu simpul dan m adalah maksimum kedalaman dari ruang status. Kompleksitas ruang keduanya juga $O(b^m)$, karena semua node perlu disimpan dalam memori untuk rekonstruksi solusi. GBFS memiliki kompleksitas yang lebih rendah secara praktis karena hanya fokus pada heuristic, namun secara teoritis juga bisa mendekati $O(b^m)$ jika heuristiknya tidak akurat. Selain itu, penggunaan struktur data priority queue menyebabkan tambahan overhead logaritmik pada setiap operasi pop dan push, yaitu O(log n). Dan untuk pengecekan visitedNode karena menggunakan HashSet sehingga kompleksitas waktunya adalah O(1). Analisis ini menunjukkan bahwa pemilihan algoritma dan heuristik yang tepat sangat krusial untuk efisiensi penyelesaian puzzle, khususnya pada papan yang lebih kompleks.

Algoritma Iterative Deepening Search (IDS) menggabungkan keunggulan BFS dan DFS dengan melakukan pencarian DFS berulang kali hingga batas kedalaman yang meningkat secara bertahap. IDDFS menjamin solusi dengan kedalaman minimum seperti BFS, namun menggunakan memori jauh lebih efisien seperti DFS karena hanya menyimpan satu jalur aktif. Kompleksitas waktu IDS adalah $O(b^d)$ dan kompleksitas ruangnya hanya O(d), dengan b adalah maksimum percabangan dari suatu simpul dan d adalah kedalaman solusi. Kelemahan utama algoritma ini adalah redundansi eksplorasi node yang berulang di tiap iterasi, membuatnya lebih lambat dari algoritma seperti A* dalam praktik. IDS cocok untuk ruang pencarian besar dan dalam dengan cost langkah yang seragam.

BAB V

IMPLEMENTASI BONUS

```
public int calcH() {
        Piece prim = null;
        for (Piece p : pieces) {
            if (p.isPrimary()) {
                prim = p;
            }
        }

        int block = 0;
        int exitX = exitGate.x;
        int exitY = exitGate.y;
        int primX = prim.getHead().x;
        int primY = prim.getHead().y;
        char primId = prim.getId();

        if (prim.isOrientationHorizontal()) {
            int dx = exitX - primX;
            if (dx > 0) {
                for (int i = 0; i < dx; i++) {
                    if (buf[primY][primX + i + 1] != '.' && buf[primY][primX +
i + 1] != primId) {
                        block++;
                    }
                }
            } else {
                dx *= -1;
                for (int i = 0; i < dx; i++) {
                    if (buf[primY][primX - i - 1] != '.' && buf[primY][primX -
i - 1] != primId) {
                        block++;
                    }
                }
            }
        } else {
            int dy = exitY - primY;
            if (dy > 0) {
                for (int i = 0; i < dy; i++) {
```

```
                    if (buf[primY + i + 1][primX] != '.' && buf[primY + i +
1][primX] != primId) {
                        block++;
                    }
                }
            } else {
                dy *= -1;
                for (int i = 0; i < dy; i++) {
                    if (buf[primY - i - 1][primX] != '.' && buf[primY - i -
1][primX] != primId) {
                        block++;
                    }
                }
            }
        }

        return block;
    }

    public int calcH2() {
        Piece prim = null;
        for (Piece p : pieces) {
            if (p.isPrimary()) {
                prim = p;
            }
        }

        int exitX = exitGate.x;
        int exitY = exitGate.y;
        int primX = prim.getHead().x;
        int primY = prim.getHead().y;
        int primSize = prim.getSize();

        if (prim.isOrientationHorizontal()) {
            int dxHead = Math.abs(exitX - primX);
            int dxTail = Math.abs(exitX - (primX + primSize - 1));
            if (dxHead < dxTail) {
                return dxHead;
            } else {
                return dxTail;
            }
```

```
        } else {
            int dyHead = Math.abs(exitY - primY);
            int dyTail = Math.abs(exitY - (primY + primSize - 1));
            if (dyHead < dyTail) {
                return dyHead;
            } else {
                return dyTail;
            }
        }
    }
```

Dua heuristik yakni jumlah sel antara posisi primary piece saat ini dengan exit gate **`calcH2()`** dan jumlah sel yang terhalang oleh piece lain antara posisi primary piece saat ini dengan exit gate **`calcH()`**. **Heuristik pertama** admissible karena untuk mencapai goal state, jarak minimum yang harus ditempuh pastilah jarak primary piece ke exit gate. Semakin jauh jarak primary piece dari exit gate, maka pastilah primary piece semakin menjauhi pintu keluar. **Heuristik kedua** admissible karena untuk mencapai goal state, jarak minimum yang harus ditempuh sebanding dengan banyak piece yang menghalangi exit gate. Semakin banyak piece yang menghalangi exit gate, maka primary piece akan semakin sulit mencapai pintu keluar. Kedua heuristic tersebut selalu meng-*underestimate* cost sebenarnya untuk mencapai goal node $h(n) \leq h^*(n)$ sehingga heuristic admissible.

```
public boolean startIDS(Node root, int maxDepth, int[] visitedNodeCtr) {
        for (int depth = 0; depth <= maxDepth; depth++) {
            visitedStates.clear();
            visitedNodes.clear();

            visitedNodes.add(root);
            visitedStates.add(root.getState().toHashString());
            visitedNodeCtr[0]++;

            if (depthLimitedSearch(root, depth, 0, visitedNodeCtr)) {
                return true;
            }
        }

        return false;
    }

    private boolean depthLimitedSearch(Node node, int limit, int nodeIndex,
```

```
int[] visitedNodeCtr) {
        visitedNodeCtr[0]++;
        if (node.isWinning()) {
            return true;
        }

        if (limit == 0) return false;

        for (Piece p : node.getState().getPieces()) {
            for (int move : node.getState().calcPossibleMove(p.getId())) {
                char pieceId = p.getId();
                Board newBoard = node.getState().clone();
                newBoard.movePiece(pieceId, move);

                String hash = newBoard.toHashString();
                if (visitedStates.contains(hash)) continue;

                Node child = node.clone();
                child.setState(newBoard);
                child.setWhat(pieceId + "-" + (p.isOrientationHorizontal() ?
                    (move > 0 ? "kanan" : "kiri") :
                    (move > 0 ? "bawah" : "atas")));

                child.getPath().add(nodeIndex);
                visitedNodes.add(child);
                int childIndex = visitedNodes.size() - 1;
                visitedStates.add(hash);

                if (depthLimitedSearch(child, limit - 1, childIndex,
visitedNodeCtr)) {
                    return true;
                }

                visitedStates.remove(hash);
                visitedNodes.remove(visitedNodes.size() - 1);
            }
        }

        return false;
    }
```

Iterative Deepening Search (IDS) adalah algoritma pencarian yang menggabungkan kelebihan dari algoritma DFS (penggunaan memori kecil) dan BFS (menemukan solusi terpendek). IDS menjalankan DFS berulang kali dengan batas kedalaman (depth limit) yang meningkat secara bertahap, dimulai dari 0 hingga mencapai solusi.

KESIMPULAN

Dalam tugas ini, telah dikembangkan sebuah program untuk menyelesaikan puzzle Rush Hour menggunakan algoritma pencarian jalur, yaitu Uniform Cost Search (UCS), Greedy Best-First Search (GBFS), A* Search, dan IDS. Dengan pendekatan pencarian berbasis priority queue dan penerapan heuristik yang sesuai ataupun IDS, program secara efisien mengeksplorasi kemungkinan langkah hingga menemukan solusi atau menyatakan bahwa solusi tidak ada.

Melalui pengujian berbagai kasus, terlihat bahwa A* Search memberikan kinerja terbaik secara keseluruhan karena mempertimbangkan baik biaya langkah (g(n)) maupun estimasi jarak ke tujuan (h(n)). Heuristik yang digunakan telah dirancang agar memenuhi kriteria admissible, serta mempercepat pencarian solusi tanpa mengorbankan akurasi. Dengan demikian, tugas ini menunjukkan bagaimana pemilihan strategi pencarian dan perancangan heuristik yang tepat dapat secara signifikan mempengaruhi performa pemecahan masalah pada puzzle Rush Hour.

SARAN

Kedepannya program ini dapat dikembangkan lebih lanjut dengan menerapkan visualisasi langkah solusi secara interaktif, serta eksplorasi algoritma lain seperti IDA* atau Bidirectional Search untuk membandingkan performa. Selain itu, pengembangan antarmuka pengguna yang lebih ramah akan sangat membantu dalam menguji berbagai skenario dan memperluas aplikasi program ini ke pengguna non-teknis.

CHECKLIST

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	✓	

PRANALA GITHUB

https://github.com/iqbalhaidr/Tucil3_13523111