Tugas Kecil III IF2211 Strategi Algoritma

Penyelesaian Puzzle *Rush Hour* Menggunakan Algoritma Pathfinding

Disusun oleh:

Adiel Rum (10123004)
Timothy Niels Ruslim (10123053)

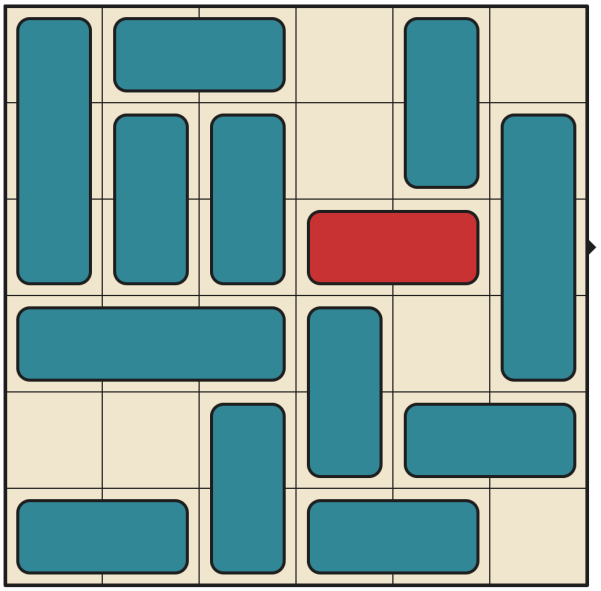
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

DAFTAR ISI

BAB I
DESKRIPSI MASALAH DAN LANDASAN TEORI

1.1 Permainan Rush Hour

Rush Hour adalah sebuah permainan puzzle logika berbasis grid yang menantang pemain untuk menggeser kendaraan di dalam sebuah kotak (biasanya berukuran 6×6) agar mobil utama (biasanya berwarna merah) dapat keluar dari kemacetan melalui pintu keluar di sisi papan. Untuk tugas ini, dilakukan perumuman grid menjadi sebarang $m \times n$, lalu setiap kendaraan direpresentasikan sebagai potongan $p \times 1$ atau $1 \times p$ tergantung orientasinya (horizontal atau vertikal) di grid.



Sumber: www.michaelfogleman.com/rush/

Setiap kendaraan hanya bisa bergerak lurus ke depan atau ke belakang sesuai dengan orientasinya, dan tidak dapat berputar. Tujuan utama dari permainan ini adalah memindahkan mobil merah ke pintu keluar. Berdasarkan ini, penyelesaian permainan Rush Hour ini bisa menjadi sangat panjang maupun sulit tergantung konfigurasi kendaraan di awal. Selebihnya, kami lebih tertarik dengan solusi dengan jumlah langkah (pemindahan kendaraan) seminimal mungkin.

1.2 Algoritma *Pathfinding*

Algoritma penentuan rute (*pathfinding*) adalah algoritma yang jika diberikan suatu ruang status dan status awal, biasa direpresentasikan dalam graf, dapat menentukan

langkah-langkah untuk mencapai status tujuan. Ini memiliki banyak penerapannya seperti di kecerdasan buatan, robotika, navigasi, dan sistem-sistem lainnya. Secara khusus, ada tiga algoritma penentuan rute yang kami soroti: *Uniform Cost Search* (UCS), *Greedy Best-First Search* (GBFS), dan *A* Search*, yang sendiri-sendiri memiliki kekuatan dan kelemahannya masing-masing. Adapun bahwa algoritma penentuan rute memiliki hubungan erat dengan heuristik. Berikut penjelasan lebih rincinya.

1.2.1. Uniform Cost Search (UCS)

Uniform Cost Search (UCS) adalah suatu algoritma *uninformed* yang melakukan pencarian node berdasarkan harga jarak terendah dari simpul awal $g(n)$. Secara teknis, UCS dapat dipandang sebagai kasus khusus dari algoritma Dijkstra. Oleh karena itu, UCS sebenarnya *complete* dan pasti menghasilkan solusi optimal. Dalam implementasinya, algoritma UCS akan membangkitkan simpul status lalu mengurutkannya dalam suatu antrian prioritas (*priority queue*) berdasarkan fungsi evaluasi

$$f(n) = g(n).$$

Jadi, pada setiap iterasinya, UCS akan selalu mengunjungi simpul dengan harga jarak terendah dari simpul awal. Ini memastikan UCS akan menghasilkan solusi optimal.

1.2.2. Greedy Best-First Search (GBFS)

Greedy Best-First Search (GBFS) adalah suatu algoritma *informed* yang melakukan pencarian node berdasarkan suatu taksiran dari harga jarak antara simpul yang akan dikunjungi dengan simpul tujuan berdasarkan suatu fungsi *heuristik* $h(n)$. GBFS jelas suatu algoritma *greedy* yang sangat agresif, sehingga solusinya tidak terjamin optimal. Lalu, dia tidak *complete*, terutama untuk ruang status tak berhingga, atau yang memiliki siklus. Dalam implementasinya, algoritma GBFS akan membangkitkan simpul status lalu mengurutkannya dalam suatu antrian prioritas (*priority queue*) berdasarkan fungsi evaluasi

$$f(n) = h(n),$$

yang murni bersifat heuristik. Oleh karena itu, kompleksitas waktunya juga akan bergantung pada heuristik yang dipilih. Walaupun GBFS tidak menjamin solusi optimal, secara umum penggunaan heuristik secara *greedy* berarti GBFS dapat mencapai suatu solusi lebih cepat.

1.2.3. A* Search

A Search* adalah suatu algoritma *informed* yang melakukan penyeimbangan antara harga untuk mencapai suatu simpul dari simpul awal $f(n)$ dan juga taksiran harga dari simpul tersebut ke simpul akhir berdasarkan suatu heuristik $g(n)$. Maka, pencarian A* menggabungkan kedua sifat dari UCS dan GBFS. Dalam implementasinya, algoritma GBFS akan membangkitkan simpul status lalu mengurutkannya dalam suatu antrian prioritas (*priority queue*) berdasarkan fungsi evaluasi

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

Dengan ini, *completeness* dan keoptimalan algoritma A* sangat tergantung heuristik yang dipilih. Jika heuristik *admissible* (tidak pernah melebihi harga sebenarnya), maka A* *complete* dan menghasilkan solusi optimal. Jika tidak, fakta tersebut tidak dijamin. Dalam kasus pertama, ini berarti A* dapat menghasilkan solusi optimal tidak seperti GBFS dan juga dengan cara yang lebih cepat daripada UCS.

1.3. Peran Heuristik

Heuristik adalah suatu teknik penggunaan aproksimasi untuk menaksir suatu harga tertentu dalam *problem-solving* secara umumnya, dan terutama pencarian rute secara khususnya. Di pencarian rute, dia menggunakan fungsi heuristik $h(n)$ yang menggunakan informasi yang dapat lebih cepat atau mudah diperoleh untuk menaksir harga sebenarnya. Misalkan $h'(n)$ adalah harga sebenarnya dari suatu simpul ke simpul tujuan. Maka, jika $h(n) \leq h'(n)$, heuristik disebut *admissible*. Untuk pencarian A*, seperti yang telah dibahas, ini dapat memastikan solusi optimal. Seni dari rekayasa algoritma adalah penentuan heuristik. Secara teknis, heuristik $h(n) = 0$ jelas *admissible*. Tetapi, penerapan ini di A* hanya akan menghasilkan UCS kembali, sehingga heuristik terlalu lemah. Di sisi lain, jika kita pilih $h(n) = h'(n)$, itu berarti kita sudah mengetahui jarak terdekat, sehingga buat apa dilakukan pencarian. Heuristik yang informatif dapat bisa menyeimbangkan kedua sisi ini, sehingga dapat mendapatkan keuntungan di sisi keoptimalan dan juga efisiensi.

BAB II
ALGORITMA PENCARIAN RUTE DI RUSH HOUR

2.1 Algoritma UCS

Langkah-langkah dari algoritma UCS dapat dituliskan sebagai *pseudocode* seperti berikut.

fungsi `UCS`

Masukan
`start` : status awal
`goal` : status tujuan

Keluaran
`path`: rute dari status awal ke status tujuan

```
q ← priority queue ordered by g(n)
q.insert(start, priority=0)
came_from ← empty map
cost_so_far ← map with start → 0

while not q.is_empty() do
   current ← q.pop()

   if current = goal then
      return RECONSTRUCT_PATH(came_from, current)

   for each neighbor in NEIGHBORS(current) do
      new_cost ← cost_so_far[current] + COST(current, neighbor)
      if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor] then
         cost_so_far[neighbor] ← new_cost
         q.insert(neighbor, priority=new_cost)
         came_from[neighbor] ← current

return failure
```

Perhatikan bahwa pemeriksaan `current = goal` khusus untuk Rush Hour akan melibatkan mencari apakah ada suatu jalur kosong antara kendaraan primary dengan pintu keluar. Dalam kata lain, perlu dicari apakah ada sel kendaraan antara piece primary dengan pintu keluar. Selebihnya, di sini $g(n)$ secara sederhana sebenarnya hanyalah kedalaman simpul yang dikunjungi di pohon pencarian, sehingga UCS sebenarnya akan sama dengan BFS di permainan Rush Hour. Seperti yang telah dijelaskan, UCS seharusnya memberikan solusi optimal.

2.2 Algoritma Greedy Best First Search

Langkah-langkah dari algoritma GBFS dapat dituliskan sebagai *pseudocode* seperti berikut.

fungsi `GBFS`

Masukan

`start` : status awal
`goal` : status tujuan

Keluaran

`path`: rute dari status awal ke status tujuan

```
q ← priority queue ordered by h(n)
q.insert(start, priority=HEURISTIC(start, goal))
came_from ← empty map
visited ← empty set

while not q.is_empty() do
   current ← q.pop()

   if current = goal then
      return RECONSTRUCT_PATH(came_from, current)

   visited.add(current)

   for each neighbor in NEIGHBORS(current) do
      if neighbor not in visited then
         q.insert(neighbor, priority=HEURISTIC(neighbor, goal))
         if neighbor not in came_from then
            came_from[neighbor] ← current

return failure
```

Perhatikan bahwa hal yang terpenting yang perlu dipikirkan hanyalah heuristiknya. Untuk permainan Rush Hour ini, suatu fungsi heuristik $h(n)$ untuk menaksir harga jarak dari suatu konfigurasi ke konfigurasi tujuan adalah jumlah kendaraan yang menghalangi jalur antara primary piece dan pintu keluar. Ini adalah heuristik yang *admissible*. Mengapa? Jika ada k kendaraan yang menghalangi jalan primary piece, ini berarti diperlukan setidaknya k langkah (atau bahkan lebih jika perlu digerakan kendaraan lain sebelum kendaraan yang menghalangi) untuk menyingkirkan mereka sebelum primary piece dapat bergerak ke pintu keluar. Jadi, $h(n)$ adalah suatu batas bawah untuk harga jarak ke simpul tujuan tersebut, yang berarti dia *admissible*. Maka, itu menjadi heuristik utama yang kami akan gunakan. Kemudian, walaupun heuristik *admissible*, keoptimalan solusi tidak dijamin (walaupun kita bisa tahu tidak akan terlalu jauh karena *admissibility* tersebut).

2.3 Algoritma A*

Langkah-langkah dari algoritma A* dapat dituliskan sebagai *pseudocode* seperti berikut.

fungsi `A*`

Masukan
`start` : status awal
`goal` : status tujuan

Keluaran
`path`: rute dari status awal ke status tujuan

```
q ← priority queue ordered by f(n) = g(n) + h(n)
q.insert(start, priority=HEURISTIC(start, goal))
came_from ← empty map
cost_so_far ← map with start → 0

while not q.is_empty() do
   current ← q.pop()

   if current = goal then
      return RECONSTRUCT_PATH(came_from, current)

   for each neighbor in NEIGHBORS(current) do
      new_cost ← cost_so_far[current] + COST(current, neighbor)
      if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor] then
         cost_so_far[neighbor] ← new_cost
         priority ← new_cost + HEURISTIC(neighbor, goal)
         q.insert(neighbor, priority)
         came_from[neighbor] ← current

return failure
```

Di sini, kita melakukan hal yang sama dengan UCS maupun GBFS tetapi dengan fungsi evaluasi jumlah

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

yaitu jumlah dari kedalaman satu status pada pohon pencarian dengan jumlah kendaraan yang menghalangi jalur primary piece ke sel keluar. Seperti yang telah dijelaskan, karena $h(n)$ *admissible*, A* akan menghasilkan solusi yang optimal dan *complete*. Secara teoritis, dia akan berjalan lebih cepat daripada UCS (walaupun mungkin lebih pelan dari GBFS), tetapi menghasilkan solusi optimal seperti UCS (tidak seperti GBFS).

BAB III
IMPLEMENTASI DAN SOURCE CODE

3.1 Implementasi Algoritma Pencarian Rute di C++

Berikut adalah implementasikan algoritma pathfinding A* untuk mencari solusi dari permainan Rush Hour dalam suatu program CLI di bahasa C++. Dipilih bahasa C++ karena merupakan bahasa pemrograman *low-level* sehingga dapat menjalankan algoritma dengan cepat. Berikut adalah modularisasi yang dilakukan.

3.1.1. Piece

Piece adalah struct yang merepresentasikan kendaraan dalam permainan rush hour, berikut adalah beberapa atribut dari *Piece*.

Atribut	Tipe Data	Deskripsi
`name`	`char`	Char/huruf yang terasosiasi dengan piece tersebut
`pos_x`	`int`	Angka yang menyatakan baris posisi piece
`pos_y`	`int`	Angka yang menyatakan kolom posisi piece
`len`	`int`	Angka yang menyatakan panjang piece
`ori`	`int`	Angka yang menyatakan orientasi dari piece, 0 jika horizontal, 1 jika vertical

3.1.2. Papan

Pertama, dibuat terlebih dahulu struct `Papan` yang berfungsi untuk menyimpan keadaan grid pada suatu state tertentu, berikut adalah beberapa atribut dari struct `Papan`.

Atribut	Tipe Data	Deskripsi
`rows`	`int`	Angka yang menyatakan banyak baris pada grid
`cols`	`int`	Angka yang menyatakan banyak kolom pada grid
`exit_x`	`int`	Angka yang menyatakan baris letak pintu keluar

exit_y	int	Angka yang menyatakan kolom letak pintu keluar
grid	vector<vector<char>>	Menyimpan bagaimana bentuk grid pada saat itu

Kemudian, papan memiliki metode berikut untuk membantu dalam pembentukan *grid* dan dalam perhitungan untuk algoritma *pathfinding*.

Metode	Parameter	Luaran	Deskripsi
Papan	vector<string>, int, int	Papan	Konstruktor yang membuat *grid*, terutama posisi dari pintu keluar dan dinding, berdasarkan masukan.
extractPieces	-	Vector<Piece>	Fungsi yang mencari semua *Piece* yang ada pada *grid*.

3.1.3. Move

Struct Move adalah struct yang merepresentasikan suatu gerakan yang dialami suatu piece, berikut adalah atribut dari Move.

Atribut	Tipe Data	Deskripsi
arah	string	String yang menyatakan arah gerak dari piece, bisa “atas”, “bawah”, “kanan”, atau “kiri”
dist	int	Angka yang menyatakan seberapa jauh piece bergerak pada orientasi tersebut

3.1.4. State

Struct State berperan sebagai data struktur yang menyimpan informasi-informasi penting pada simpul pencarian. Berikut adalah beberapa atribut dari State.

Atribut	Tipe Data	Deskripsi
Papan	Papan	Menyimpan informasi kondisi *Papan* pada simpul
pieces	vector<Piece>	Vektor yang menyimpan semua piece yang ada pada papan
list_moves	vector<pair<Piece, Move>>	Vektor yang berisi gerakan-gerakan yang sudah diterapkan pada simpul-simpul sebelumnya, berbentuk pair yang berisi *Piece*

		dan *Move*, masing-masing merepresentasikan piece yang digerakkan ke arah mana dan seberapa banyak
`depth`	`int`	Angka yang menunjukkan kedalaman simpul pada pohon pembangkitan pencarian
`row`	`int`	Angka menunjukkan banyak baris pada grid
`col`	`int`	Angka menunjukkan banyak kolom pada grid

Metode pada `State` adalah sebagai berikut.

Metode	Parameter	Luaran	Deskripsi
`getAllPossibleMoves`	-	`vector<pair<Piece, Move>>`	Vektor yang berisi semua gerakan yang mungkin dari semua piece yang ada
`applyMove`	`pair<Piece, Move>`	`State`	Menerapkan perpindahan *Piece* sebesar *Move* pada *State*
`computeDistance`	-	`int`	Menghitung jarak primary piece ke exit pada *State* tersebut
`computeBlocls`	-	`int`	Menghitung banyak piece yang menghalangi Primary piece ke exit
`computeEuclideanDistance`		`double`	Menghitung jarak euclid primary piece ke exit

3.1.5. Solver

Struct `Solver` berperan sebagai data struktur yang menyimpan informasi-informasi penting pada simpul pencarian. Berikut adalah beberapa atribut dari `Solver`.

Atribut	Tipe Data	Deskripsi
`type`	`int`	Angka yang merepresentasikan jenis algoritma yang digunakan; dengan 0: UCS, 1: GBFS, 2: A*.

Metode pada `Solver` adalah sebagai berikut.

Metode	Parameter	Luaran	Deskripsi
`gridToString`	`vector<vector<char>>`	`string`	Mengkonversikan *grid* menjadi suatu string; digunakan untuk mengecek apakah suatu

			konfigurasi pernah ditelusuri.
`toggleType`	`int`	`pair<int, int>`	Mengkonversikan `type` menjadi suatu parameter biner (a, b) yang dapat digunakan untuk menghitung fungsi evaluasi $f(n)$.
`solveBoard`	`State`	`vector<pair<Piece, Move>>`	Mencari solusi dari status awal dari papan menggunakan algoritma di `type`. Digunakan implementasi *priority queue* berdasarkan fungsi evaluasi $f(n)$. Lalu, pencarian diberhentikan saat simpul tujuan telah ditemukan (yang berarti sudah ada jalur kosong untuk *primary piece* menuju pintu keluar).

3.1.6. Main

Program CLI yang mengintegrasikan algoritma kompresi gambar ini dengan interface ada pada `App.java`. Dia tidak ada atribut dan hanya memiliki dua metode.

Metode	Parameter	Luaran	Deskripsi
`main`	-	-	Fungsi yang menjadi *entry point* dalam eksekusi program C++. Di sini, ini berarti dia adalah program CLI Rush Hour Solver kami.

Secara teknis, berikut cara kerja `main` di `Main`. Pertama, program meminta masukan berupa hal-hal berikut.

1. Alamat suatu file .txt yang memuat konfigurasi permainan Rush Hour. File tersebut terdiri atas beberapa hal berikut.
    a. Dimensi papan yaitu $A \times B$.
    b. Banyak *pieces* (kendaraan) yang **bukan primary** sebanyak N.
    c. Konfigurasi papan yang berupa penempatan primary piece, non-primary piece, dan lokasi *pintu keluar*. Primary piece menggunakan huruf **P** dan lokasi pintu keluar menggunakan huruf **K**. Piece-piece lainnya menggunakan huruf selain itu dua. Sel kosong diisi dengan '.' (titik).
2. Algoritma pencarian rute (*pathfinding*) yang digunakan berupa integer seperti berikut.
    a. `0`: Uniform Cost Search (UCS)

b. `1`: Greedy Best First Search (GBFS)

c. `2`: A* Search

Kemudian, program akan mencoba melakukan kompresi berdasarkan parameter dan gambar yang dimasukan oleh pengguna. Jika selesai, program akan memberikan keluaran berikut.

1. Jumlah status konfigurasi (simpul) yang dikunjungi, yaitu jumlah *gerakan* yang dicoba.
2. Waktu eksekusi pencarian solusi (dalam ms).
3. Konfigurasi papan pada setiap langkah pergerakan dari status awal ke status tujuan. Konfigurasi tersebut di-*print* di console menggunakan warna, dengan merah adalah piece primary, hijau adalah sel pintu keluar, dan kuning adalah piece yang digerakkan pada iterasi tersebut.

Terakhir, adapun bahwa program ini berjalan dalam suatu while loop, sehingga setelah program selesai memberikan solusi, dia dapat kembali menerima input untuk mencari solusi untuk permainan berikutnya.

3.2 Source Code Program

Berikut adalah *source code* dari program di bahasa C++ dengan menerapkan semua yang telah dijelaskan di atas.

3.2.1. Piece

```
#ifndef STATE_H
#define STATE_H

#include <bits/stdc++.h>
#include "papan.h"
#include "move.h"

using namespace std;

struct State {
   Papan papan;
   vector<Piece> pieces;
   vector<pair<Piece, Move>> list_moves;
   int depth;
   int row, col;

   // Konstruktor
   State(
      const Papan& _papan,
      const vector<pair<Piece,Move>>& _list_moves,
      const vector<Piece>& _pieces,
      int _depth
   );

   // Metoda
   vector<pair<Piece, Move>> getAllPossibleMoves();
```

```
   void printPieces();
   void printAllPossibleMoves();
   void printAllMoves();
   State applyMove(const pair<Piece, Move>& action) const;
   int computeDistance() const;
   int computeBlocks() const;
   int computeDoubleBlocks() const;
   double computeEuclideanDistance() const;

};

#endif
```

```
#include "piece.h"
using namespace std;

// Konstruktor
Piece::Piece() : name(' '), pos_x(0), pos_y(0), len(0), ori(0) {}
Piece::Piece(int _pos_x, int _pos_y, int _len, int _ori, char _name)
    : pos_x(_pos_x), pos_y(_pos_y), len(_len), ori(_ori), name(_name) {}

// Display piece
void Piece::displayPiece() {
    cout << "huruf: " << name
         << " baris: " << pos_x
         << " kolom: " << pos_y
         << " panjang: " << len
         << " orientasi: " << ((ori == 0) ? "horizontal" : "vertical")
         << endl;
}
```

3.2.2. Papan

```
#ifndef PAPAN_H
#define PAPAN_H

#include <bits/stdc++.h>
#include "Piece.h"

using namespace std;

struct Papan {
    int rows;
    int cols;
    int exit_x;
    int exit_y;
    bool is_valid;
    vector<vector<char>> grid;

    // Constructor
    Papan(vector<string> _grid, int _rows, int _cols);

    // Method
    void printGrid(char s = '-');
    vector<Piece> extractPieces();
};

#endif
```

```
#include "papan.h"

// Konstruktor
Papan::Papan(vector<string> _grid, int _rows, int _cols)
    : rows(_rows + 2), cols(_cols + 2), exit_x(0), exit_y(0) {
```

```
    // Initialize grid with sentinels
    grid.resize(rows, vector<char>(cols, '*'));

    // Find position of exit cell (K)
    int raw_exit_x = -1;
    int raw_exit_y = -1;
    bool empty_row = false;
    int row_delete = -1;
    int col_delete = -1;

    for (int i = 0; i < _grid.size(); i++) {

        // Locate exit cell
        size_t pos = _grid[i].find('K');
        if (pos != std::string::npos) {

            // Save exit cell coordinates
            raw_exit_x = i;
            raw_exit_y = static_cast<int>(pos);

            // Check if top or bottom exit
            int non_empty_count = 0;
            for (int j = 0; j < _grid[i].size(); ++j) {
                if (j == raw_exit_y) continue;
                if (_grid[i][j] != ' ') non_empty_count++;
            }
            if (non_empty_count == 0) {
                empty_row = true;
            }

            // Erase exit cell
            _grid[i].erase(pos, 1);
            if (i == 0 || i == _rows) {
                row_delete = raw_exit_x;
            }
            if (pos == 0 || pos == _cols) {
                col_delete = raw_exit_y;
            }

            // Check if valid exit
            is_valid = true;
            bool found_primary = false;

            // Exit at top or bottom row
            if ((raw_exit_x == 0 && empty_row) || raw_exit_x == _rows) {
                for (int i = 0; i < _rows - 1; ++i) {
                    // Check vertical primary piece
                    if (_grid[i][raw_exit_y] == 'P' && _grid[i + 1][raw_exit_y] == 'P')
{
                        found_primary = true;
                        break;
                    }
                }
            }

            // Exit at left or right column
            else {
                // Check horizontal primary piece
                string& row = _grid[raw_exit_x];
                if (row.find("PP") != string::npos) {
                    found_primary = true;
                }
            }

            // Early return if invalid
            if (!found_primary) {
                is_valid = false;
```

```
                // return;
            }

            // Add padding
            exit_x = -1;
            exit_y = -1;

            // Top row
            if (raw_exit_x == 0) {
                if (empty_row) {
                    exit_x = raw_exit_x;
                    exit_y = raw_exit_y + 1;
                } else {
                    if (raw_exit_y == 0) {
                        exit_x = raw_exit_x + 1;
                        exit_y = raw_exit_y;
                    } else {
                        exit_x = raw_exit_x + 1;
                        exit_y = raw_exit_y + 1;
                    }
                }
            }

            // Bottom row
            else if (raw_exit_x == _rows) {
                exit_x = raw_exit_x + 1;
                exit_y = raw_exit_y + 1;
            }

            // Left column
            else if (raw_exit_y == 0) {
                exit_x = raw_exit_x + 1;
                exit_y = raw_exit_y;
            }

            // Right column
            else if (raw_exit_y == _cols) {
                exit_x = raw_exit_x + 1;
                exit_y = raw_exit_y + 1;
            }

            // Invalid
            else {
                is_valid = false;
                return;
            }

            break;
        }
    }

    // Remove exit row if necessary
    if (row_delete != -1) {
        _grid.erase(_grid.begin() + row_delete);
    }

    // Remove exit column if necessary
    if (col_delete == 0) {
        for (string& row : _grid) {
            if (!row.empty() && isspace(row[0])) {
                row.erase(0, 1);  // Remove whitespace
            }
        }
    }

    // Copy grid to papan
    for(int i = 0; i < _rows; i++) {
```

```
        for(int j = 0; j < _cols; j++) {
            grid[i+1][j+1] = _grid[i][j];
        }
    }

    // Open exit cell
    grid[exit_x][exit_y] = 'K';
}

// Display papan
void Papan::printGrid(char s) {
    for (const auto& row : grid) {
        for (char c : row) {
            if (c == s) {
                cout << "\033[43m" << c << "\033[0m ";
            } else if (c == 'P') {
                cout << "\033[1;31m" << c << "\033[0m ";
            } else if (c == 'K') {
                cout << "\033[1;32m" << c << "\033[0m ";
            } else if (c == '*') {
                cout << "\033[1;37m" << c << "\033[0m ";
            } else {
                cout << c << "\033[0m ";
            }
        }
        cout << '\n';
    }
}

// Get all pieces from papan
vector<Piece> Papan::extractPieces() {
    unordered_set<char> visited;
    vector<Piece> pieces;

    // Loop through entire grid
    int n = grid.size(), m = grid[0].size();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            char ch = grid[i][j];

            // Skip empty or sentinel cells and visited pieces
            if (ch == '.' || ch == '*' || ch == 'K' || visited.count(ch)) continue;

            int len = 1, ori = -1;

            // Horizontal piece
            if (j + 1 < m && grid[i][j + 1] == ch) {
                ori = 0;
                int jj = j + 1;
                while (jj < m && grid[i][jj] == ch) {
                    ++len; ++jj;
                }
            }

            // Vertical piece
            else if (i + 1 < n && grid[i + 1][j] == ch) {
                ori = 1;
                int ii = i + 1;
                while (ii < n && grid[ii][j] == ch) {
                    ++len; ++ii;
                }
            }

            // Add piece to list and mark as visited
            if (ori != -1) {
                pieces.emplace_back(i, j, len, ori, ch);
                visited.insert(ch);
```

```
            }
        }
    }

    return pieces;
}
```

3.2.3. Move

```
#ifndef MOVE_H
#define MOVE_H

#include <bits/stdc++.h>
using namespace std;

struct Move {
    string arah; // arah gerak: "atas", "bawah", "kanan", "kiri"
    int dist;    // jarak gerak

    // Constructor
    Move();
    Move(string _arah, int _dist);

    // Method
    void displayMove();
};

#endif
```

```
#include "move.h"

Move::Move() : arah(" "), dist(0) {}

Move::Move(string _arah, int _dist)
    : arah(_arah), dist(_dist) {}

// Display move
void Move::displayMove() {
    cout << "arah: " << arah << " sebanyak " << dist << endl;
}
```

3.2.4. State

```
#ifndef STATE_H
#define STATE_H

#include <bits/stdc++.h>
#include "papan.h"
#include "move.h"

using namespace std;

struct State {
   Papan papan;
   vector<Piece> pieces;
   vector<pair<Piece, Move>> list_moves;
   int depth;
   int row, col;

   // Konstruktor
   State(
      const Papan& _papan,
      const vector<pair<Piece,Move>>& _list_moves,
```

```
      const vector<Piece>& _pieces,
      int _depth
   );

   // Metoda
   vector<pair<Piece, Move>> getAllPossibleMoves();
   void printPieces();
   void printAllPossibleMoves();
   void printAllMoves();
   State applyMove(const pair<Piece, Move>& action) const;
   int computeDistance() const;
   int computeBlocks() const;
   double computeEuclideanDistance() const;

};

#endif
```

```
#include "state.h"

// Constructor
State::State(const Papan& _papan, const vector<pair<Piece, Move>>& _list_moves, const
vector<Piece>& _pieces, int _depth)
   : papan(_papan), list_moves(_list_moves), pieces(_pieces), depth(_depth),
row(_papan.rows), col(_papan.cols) {}

// Get all possible moves
vector<pair<Piece, Move>> State::getAllPossibleMoves() {
   vector<pair<Piece, Move>> moves;

   for (Piece& piece : pieces) {
      int x = piece.pos_x;
      int y = piece.pos_y;

      // Horizontal piece
      if (piece.ori == 0) {

         // Attmpt to move left
         int steps = 1;
         while (papan.grid[x][y - steps] == '.') {
            moves.emplace_back(piece, Move("kiri", steps));
            steps++;
         }

         // Attempt to move right
         steps = 1;
         int tail = y + piece.len - 1;
         while (papan.grid[x][tail + steps] == '.') {
            moves.emplace_back(piece, Move("kanan", steps));
            steps++;
         }

      }

      // Vertical piece
      else {

         // Attempt to move up
         int steps = 1;
         while (papan.grid[x - steps][y] == '.') {
            moves.emplace_back(piece, Move("atas", steps));
            steps++;
         }

         // Attempt to move down
         steps = 1;
         int tail = x + piece.len - 1;
```

```
         while (papan.grid[tail + steps][y] == '.') {
            moves.emplace_back(piece, Move("bawah", steps));
            steps++;
         }
      }
   }

   return moves;
}

// Print pieces
void State::printPieces() {
   for (const auto& p : pieces) {
      cout << "Piece " << p.name << " at (" << p.pos_x << ", " << p.pos_y << "), len="
<< p.len
           << ", ori=" << (p.ori == 0 ? "horizontal" : "vertical") << endl;
   }
}

// Print all possible moves
void State::printAllPossibleMoves() {
   vector<pair<Piece, Move>> moves = getAllPossibleMoves();
   for (auto& a : moves) {
      a.first.displayPiece();
      a.second.displayMove();
   }
}

// Print all moves made so far
void State::printAllMoves() {
   for (auto& a : list_moves) {
      a.first.displayPiece();
      a.second.displayMove();
   }
}

State State::applyMove(const pair<Piece, Move>& action) const {
   Piece target = action.first;
   Move move     = action.second;

   // Copy current state
   vector<pair<Piece, Move>> current_moves = this->list_moves;
   vector<Piece> new_pieces = this->pieces;
   Papan new_papan = this->papan;

   // Find & update the moved piece
   for (Piece& p : new_pieces) {
      if (p.name != target.name) continue;

      // 1) Erase from its old cells
      if (p.ori == 0) {  // horizontal
         for (int j = 0; j < p.len; ++j)
            new_papan.grid[p.pos_x][p.pos_y + j] = '.';
      } else {          // vertical
         for (int i = 0; i < p.len; ++i)
            new_papan.grid[p.pos_x + i][p.pos_y] = '.';
      }

      // 2) Update its coords
      if      (move.arah == "kiri")  p.pos_y -= move.dist;
      else if (move.arah == "kanan") p.pos_y += move.dist;
      else if (move.arah == "atas")  p.pos_x -= move.dist;
      else if (move.arah == "bawah") p.pos_x += move.dist;

   // 3) Detect exit move (only for 'P'), using original piece to find exit side:
   bool isExitMove = false;
   if (p.name == 'P') {
```

```
      // original piece before move:
      const Piece& orig = action.first;
      int o_head_x = orig.pos_x;
      int o_head_y = orig.pos_y;
      int o_tail_x = orig.pos_x + (orig.ori == 1 ? orig.len - 1 : 0);
      int o_tail_y = orig.pos_y + (orig.ori == 0 ? orig.len - 1 : 0);

      int ex = new_papan.exit_x;
      int ey = new_papan.exit_y;

      // decide exit direction once:
      enum { RIGHT, LEFT, DOWN, UP, NONE } exitDir = NONE;
      if (orig.ori == 0 && o_head_x == ex) {
         if (ey > o_tail_y) exitDir = RIGHT;
         else if (ey < o_head_y) exitDir = LEFT;
      }
      else if (orig.ori == 1 && o_head_y == ey) {
         if (ex > o_tail_x) exitDir = DOWN;
         else if (ex < o_head_x) exitDir = UP;
      }

      // now see if this move matches that direction & crosses the exit
      int head_x = p.pos_x;
      int head_y = p.pos_y;
      int tail_x = p.pos_x + (p.ori == 1 ? p.len - 1 : 0);
      int tail_y = p.pos_y + (p.ori == 0 ? p.len - 1 : 0);

      switch (exitDir) {
         case RIGHT:
         if (move.arah == "kanan" && tail_y >= ey) isExitMove = true;
         break;
         case LEFT:
         if (move.arah == "kiri"  && head_y <= ey) isExitMove = true;
         break;
         case DOWN:
         if (move.arah == "bawah" && tail_x >= ex) isExitMove = true;
         break;
         case UP:
         if (move.arah == "atas"  && head_x <= ex) isExitMove = true;
         break;
         default:
         break;
      }
   }

      // 4) Redraw unless it's the exit move
      if (!isExitMove) {
         if (p.ori == 0) {
            for (int j = 0; j < p.len; ++j)
               new_papan.grid[p.pos_x][p.pos_y + j] = p.name;
         } else {
            for (int i = 0; i < p.len; ++i)
               new_papan.grid[p.pos_x + i][p.pos_y] = p.name;
         }
      }

      // 5) Record the move
      current_moves.emplace_back(p, move);
      break;
   }

   return State(new_papan, current_moves, new_pieces, this->depth + 1);
}

// Heuristic 1 (Admissible): Calculate blocking cars
int State::computeBlocks() const {
```

```
   vector<vector<char>> grid = papan.grid;
   pair<int, int> k_pos = {papan.exit_x, papan.exit_y};
   int blocked_cells = 0;

   // Find primary piece
   for (Piece p : pieces) {
      if (p.name == 'P') {

         // Horizontal primary piece
         if (p.ori == 0) {

            // Coordinate bounds
            int i_0 = min(p.pos_y, k_pos.second);
            int i_a = max(p.pos_y, k_pos.second);

            // Count blocked cells between primary cell and exit cell
            for (int i = i_0; i < i_a; i++) {
               char c = grid[p.pos_x][i];
               if (c != 'K' && c != 'P' && c != '.') {
                  ++blocked_cells;
               }
            }

         }

         // Vertical primary piece
         else {

            // Coordinate bounds
            int i_0 = min(p.pos_x, k_pos.first);
            int i_a = max(p.pos_x, k_pos.first);

            // Count blocked cells between primary cell and exit cell
            for (int i = i_0; i < i_a; i++) {
               char c = grid[i][p.pos_y];
               if (c != 'K' && c != 'P' && c != '.') {
                  ++blocked_cells;
               }
            }
         }

         break;

      }
   }

   return blocked_cells;

}

// Heuristic 2 (Non-admissible): Compute distance between primary piece and exit cell
int State::computeDistance() const {
   vector<vector<char>> grid = papan.grid;
   pair<int, int> p_edge;
   pair<int, int> k_pos = {papan.exit_x, papan.exit_y};

   // Find primary piece edge cell
   for (Piece p : pieces) {
      if (p.name == 'P') {

         // Bottom-right-most cell
         if (k_pos.first != 0 && k_pos.second != 0) {

            // Horizontal → right-most cell
            if (p.ori == 0) {
               p_edge = {p.pos_x, p.pos_y + p.len - 1};
            }
```

```
            // Vertical → bottom-most cell
            else {
               p_edge = {p.pos_x + p.len - 1, p.pos_y};
            }

            break;
         }

          // Top-left-most cell
         p_edge = {p.pos_x, p.pos_y};
         break;

      }
   }

   // Return shortest distance
   return abs(p_edge.first - k_pos.first) + abs(p_edge.second - k_pos.second);
}
```

3.2.5. Solver

```
#ifndef SOLVER_H
#define SOLVER_H

#include <bits/stdc++.h>
#include "piece.h"
#include "move.h"
#include "papan.h"
#include "state.h"

struct Solver {
   int type; // (0: UCS, 1: Greedy, 2: A*)

   // Konstruktor
   Solver(int _type);

   // Helper
   string gridToString(const vector<vector<char>>& grid);
   pair<int, int> toggleType(int type);

   // Metoda
   vector<pair<Piece, Move>> solveBoard(const State& initial_state);

};

#endif
```

```
#include <bits/stdc++.h>
#include "solver.h"

// Constructor
Solver::Solver(int _type)
   : type(_type) {}

// Convert grid to string
string Solver::gridToString(const vector<vector<char>>& grid) {
   string result;

   // Skip the sentinel border (first and last rows/columns)
   for (size_t i = 1; i < grid.size() - 1; i++) {
      for (size_t j = 1; j < grid[i].size() - 1; j++) {
         result.push_back(grid[i][j]);
      }
   }
```

```
   return result;
}

// Convert algorithm type to parameter toggle
pair<int, int> Solver::toggleType(int type) {

   // UCS
   if (type == 0) {
      return {1,0};
   }

   // Greedy
   else if (type == 1) {
      return {0,1};
   }

   // A*
   else {
      return {1,1};
   }

}

// Solve the board with specified algorithm
vector<pair<Piece, Move>> Solver::solveBoard(const State& initial_state) {
   int nodes_explored = 0;

   // Create priority queue
   auto compare = [](const pair<int, State>& a, const pair<int, State>& b) {
      return a.first > b.first;  // Lower cost has higher priority
   };
   priority_queue<pair<int, State>, vector<pair<int, State>>, decltype(compare)>
pq(compare);

   // Initialize visited states array
   unordered_set<string> visited;

   // Calculate initial cost: f(0) = g(0) + h(0)
   pair<int, int> params = toggleType(type);
   int initial_cost = initial_state.depth * params.first + initial_state.computeBlocks()
* params.second;

   // Push initial state
   pq.push({initial_cost, initial_state});

   // Loop through priority queue
   while (!pq.empty()) {

      // Get state with lowest cost
      State current = pq.top().second;
      pq.pop();

      // Skip state if already visited
      string grid_str = gridToString(current.papan.grid);
      if (visited.count(grid_str)) continue;

      // Add to visited
      visited.insert(grid_str);
      nodes_explored++;

      // Find the primary piece (P)
      Piece* p_piece = nullptr;
      for (const Piece& p : current.pieces) {
         if (p.name == 'P') {
            p_piece = const_cast<Piece*>(&p);
            break;
```

```
        }
    }

    // Check if goal state is reached (primary piece can exit)
    int px = p_piece->pos_x;
    int py = p_piece->pos_y;
    int ori = p_piece->ori;
    int len = p_piece->len;
    int tail;
    bool can_exit = false;

    // Horizontal car
    if (ori == 0) {
        tail = py + len - 1;
        if (px == current.papan.exit_x) { // ensure same row

            // Exit is to the right
            if (current.papan.exit_y > tail) {
                can_exit = true;
                for (int c = tail + 1; c <= current.papan.exit_y; c++) {
                    char cell = current.papan.grid[px][c];
                    if (cell != '.' && cell != 'K') { can_exit = false; break; }
                }
            }

            // Exit is to the left
            else if (current.papan.exit_y < py) {
                can_exit = true;
                for (int c = py - 1; c >= current.papan.exit_y; c--) {
                    char cell = current.papan.grid[px][c];
                    if (cell != '.' && cell != 'K') { can_exit = false; break; }
                }
            }

        }
    }

    // Vertical car
    else {
        tail = px + p_piece->len - 1;
        if (py == current.papan.exit_y) { // ensure same column

                // Exit is below
                if (current.papan.exit_x > tail) {
                    can_exit = true;
                    for (int r = tail + 1; r <= current.papan.exit_x; r++) {
                        char cell = current.papan.grid[r][py];
                        if (cell != '.' && cell != 'K') { can_exit = false; break; }
                    }
                }

                // Exit is above
                else if (current.papan.exit_x < px) {
                    can_exit = true;
                    for (int r = px - 1; r >= current.papan.exit_x; r--) {
                        char cell = current.papan.grid[r][py];
                        if (cell != '.' && cell != 'K') { can_exit = false; break; }
                    }
                }

        }
    }

    // Goal state reached
    if (can_exit) {

        // 1. Figure out how far P must go to exit
```

```
         int tail = (p_piece->ori == 0)
               ? p_piece->pos_y + p_piece->len - 1
               : p_piece->pos_x + p_piece->len - 1;

         int dist_to_exit = 0;
         if (p_piece->ori == 0) {
               // horizontal
               if (current.papan.exit_y > tail) {
                  dist_to_exit = current.papan.exit_y - tail;
               } else {
                  // exit to the left
                  dist_to_exit = p_piece->pos_y - current.papan.exit_y;
               }
         } else {
               // vertical
               if (current.papan.exit_x > tail) {
                  dist_to_exit = current.papan.exit_x - tail;
               } else {
                  // exit above
                  dist_to_exit = p_piece->pos_x - current.papan.exit_x;
               }
         }

         // 2. Append the final move
         current.list_moves.emplace_back(*p_piece,
            Move(
               (p_piece->ori == 0)
                  ? (current.papan.exit_y > tail ? "kanan" : "kiri")
                  : (current.papan.exit_x > tail ? "bawah" : "atas"),
               dist_to_exit + p_piece->len
            )
         );

         cout << "Solusi ditemukan!" << endl;
         cout << "Dikunjungi " << nodes_explored << " simpul dalam ";
         return current.list_moves;
      }

      // Generate all possible next states
      vector<pair<Piece, Move>> possible_moves = current.getAllPossibleMoves();
      for (const auto& move : possible_moves) {

         // Create new state by applying the move
         State next_state = current.applyMove(move);

         // Skip already visited states
         string next_grid_str = gridToString(next_state.papan.grid);
         if (visited.count(next_grid_str)) continue;

         // Calculate cost: f(n) = g(n) + h(n)
         int cost = next_state.depth * params.first + next_state.computeBlocks() *
params.second;

         // Add to priority queue
         pq.push({cost, next_state});

      }
   }

   cout << "Solusi tidak ditemukan." << endl;
   cout << "Dikunjungi " << nodes_explored << " simpul dalam ";
   return {};  // No solution found
}
```

3.2.6. Main

```
#include <bits/stdc++.h>
#include "piece.h"
#include "move.h"
#include "papan.h"
#include "state.h"
#include "solver.h"

using namespace std;
using namespace chrono;

bool isInteger(const string& s) {
    try {
        size_t idx;
        stoi(s, &idx);
        return idx == s.size(); // ensure whole string was consumed
    } catch (invalid_argument& e) {
        return false;
    } catch (out_of_range& e) {
        return false;
    }
}

int main()
{
    cout <<
"\n=============================================================================================
====" << endl;
    cout << R"(
    _____                _       _     _                          _____          _
    |  __ \              | |     | |   | |                        / ____|        | |
    | |__) |   _ ___| |__   | |__| | ___  _   _ _ __   | (___   ___ | |_   _____ _ __
    |  _  / | | / __| '_ \  |  __  |/ _ \| | | | '__|  \___ \ / _ \| \ \ / / _ \ '__|
    | | \ \ |_| \__ \ | | | | |  | | (_) | |_| | |     ____) | (_) | |\ V /  __/ |
    |_|  \_\__,_|___/_| |_| |_|  |_|\___/ \__,_|_|    |_____/ \___/|_| \_/ \___|_|
    )" << endl;
    cout <<
"=============================================================================================
==\n" << endl;

    while (true) {

        // Input file
        string file_path;
        ifstream input_file(file_path);
        do {
            cout << "Masukan file (.txt): ";
            getline(cin, file_path);
            input_file.close();
            input_file.open(file_path);
            if (!input_file.is_open()) {
                cout << "\nFile tidak bisa dibuka. Coba lagi. \n";
            }
        } while (!input_file.is_open());
        cout << endl;

        // Read file
        int N, M, num_piece;
        input_file >> N >> M >> num_piece;

        vector<string> temp_board;
        string temp_row;
        input_file.ignore(numeric_limits<streamsize>::max(), '\n'); // skip empty line
        while(getline(input_file, temp_row)){
            temp_board.push_back(temp_row);
        }
```

```
        // Create papan
        Papan board = Papan(temp_board, N, M);
        if (!board.is_valid) {
            cout << "Papan tidak valid. Coba lagi.\n";
            continue;
        }

        // Input algoritma
        string algoritma_type;
        cout << "Berikut beberapa algoritma pathfinding." << endl;
        cout << "  0. Uniform Cost Search (UCS)" << endl;
        cout << "  1. Greedy Best First Search" << endl;
        cout << "  2. A* Search" << endl;
        cout << "Pilih algoritma (0,1,2): ";
        while(getline(cin, algoritma_type)) {
            if (algoritma_type == "0" || algoritma_type == "1" || algoritma_type == "2")
{
                break;
            }
            cout << "\nAngka tidak valid. Coba lagi. " << endl;
            cout << "Pilih algoritma (0,1,2): ";
        }

        // State awal
        cout << "\nPapan Awal" << endl;
        board.printGrid();
        cout << endl;
        vector<Piece> pieces = board.extractPieces();
        vector<pair<Piece, Move>> moves;
        State current_state(board, moves, pieces, 0);

        // Solve papan
        Solver boardSolver = Solver(stoi(algoritma_type));
        auto start = high_resolution_clock::now();
        vector<pair<Piece, Move>> solution = boardSolver.solveBoard(current_state);

        // Count time
        auto end = high_resolution_clock::now();
        auto duration = duration_cast<microseconds>(end - start);
        double time = duration.count() / 1000.0;
        cout << fixed << setprecision(3) << time << " ms. \n" << endl;

        // Print solution
        if (!solution.empty()) {
            for (size_t i = 0; i < solution.size(); i++) {

                pair<Piece, Move> frame = solution[i];

                // Print move information
                cout << "Gerakan " << (i+1) << ": ";
                cout << "Pindah piece " << frame.first.name << " ke ";
                cout << frame.second.arah << " " << frame.second.dist << " langkah." <<
endl;

                // Print new state
                current_state = current_state.applyMove(solution[i]);
                current_state.papan.printGrid(solution[i].first.name);
                cout << endl;
            }
        }

        cout <<
"================================================================================
==\n" << endl;

    }
```

```
}
```

3.2.7. (*BONUS*) GUI

Adapun bahwa dibuat GUI untuk program ini menggunakan *Python* dengan library *tkinter*.

```
import tkinter as tk
from tkinter import scrolledtext, messagebox, filedialog
import subprocess
import threading
import os
import re

class RushHourGUI(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Rush Hour Solver")
        self.geometry("800x800")

        # Input area
        tk.Label(self, text="Enter Algorithm (0: UCS, 1: GBFS, 2:
A*):").pack(anchor='nw', padx=10, pady=5)
        self.string_input = tk.Entry(self, width=50)
        self.string_input.pack(padx=10, pady=(0, 10))

        tk.Label(self, text="Enter path to .txt file (or use
Browse):").pack(anchor='nw', padx=10, pady=(5, 0))
        frame = tk.Frame(self)
        frame.pack(padx=10, pady=(0, 5), fill='x')

        self.file_path_entry = tk.Entry(frame, width=40)
        self.file_path_entry.pack(side='left', expand=True, fill='x')

        browse_btn = tk.Button(frame, text="Browse", command=self.browse_file)
        browse_btn.pack(side='left', padx=(5, 0))

        load_btn = tk.Button(frame, text="Load File", command=self.load_file)
        load_btn.pack(side='left', padx=(5, 0))

        # Puzzle
        tk.Label(self, text="Enter puzzle (N M, num_pieces, then N rows; optional extra
K-row):").pack(anchor='nw', padx=10, pady=5)
        self.input_text = scrolledtext.ScrolledText(self, width=50, height=10)
        self.input_text.pack(padx=10)
        self.input_text.insert(tk.END, "6
6\n11\nAAB..F\n..BCDF\nGPPCDFK\nGH.III\nGHJ...\nLLJMM.")

        self.solve_btn = tk.Button(self, text="Solve", command=self.solve)
        self.solve_btn.pack(pady=10)

        # Canvas
        self.cell_size = 40
        self.canvas = tk.Canvas(self, bg='white')
        self.canvas.pack(expand=True, fill='both', padx=10, pady=10)

        self.status = tk.Label(self, text="Status: waiting for input...")
        self.status.pack(anchor='sw', padx=10, pady=5)

        # Data
        self.raw = []
        self.grid = []
        self.moves = []
        self.piece_items = {}
        self.nodes = None
        self.duration = None

    def browse_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Text Files", "*.txt")])
        if file_path:
            self.file_path_entry.delete(0, tk.END)
```

```
            self.file_path_entry.insert(0, file_path)

    def load_file(self):
        path = self.file_path_entry.get().strip()
        if not path or not os.path.isfile(path):
            messagebox.showerror("Error", "Invalid file path.")
            return
        try:
            with open(path, 'r') as f:
                content = f.read()
            self.input_text.delete("1.0", tk.END)
            self.input_text.insert(tk.END, content)
        except Exception as e:
            messagebox.showerror("Error", f"Failed to read file:\n{e}")

    def solve(self):
        string_part = self.string_input.get().strip()
        input_data = self.input_text.get("1.0", tk.END).strip()

        if not input_data:
            messagebox.showerror("Error", "Please enter puzzle input.")
            return

        # Combine the inputs — modify this logic to suit your format
        combined_input = string_part + "\n" + input_data if string_part else input_data

        self.solve_btn.config(state=tk.DISABLED)
        self.status.config(text="Status: solving...")
        threading.Thread(target=self.run_solver, args=(combined_input,)).start()

    def run_solver(self, input_data):
        # 1) Read & sanitize raw lines
        lines = input_data.splitlines()
        # print(lines)
        algo = map(int, lines[0].split())
        N, M = map(int, lines[1].split())
        raw = lines[3:3+N]
        # if no K found, grab the extra line
        if (all('K' not in row for row in raw) and len(lines) > 3+N) or ('K' in raw[0]):
            raw.append(lines[3+N])

        def print_Grid(Grid):
            for i in range(len(Grid)):
                for j in range(len(Grid[i])):
                    print(Grid[i][j],end=" ")
                print()

        # print_Grid(raw)
        # 2) Locate raw K cell and detect empty-row condition
        raw_exit_x = raw_exit_y = None
        empty_row = False
        for i, row in enumerate(raw):
            if 'K' in row:
                raw_exit_x, raw_exit_y = i, row.index('K')
                if(raw_exit_y == 0):
                    for j, rew in enumerate(raw):
                        if i != j:
                            rews = rew.replace(' ', '', 1)
                            raw[j] = rews
                row = row.replace('K', '', 1)
                raw[i] = row
                # print(raw_exit_x)
                # check if every other character is space (or '.')
                if raw_exit_x == 0 or raw_exit_y == len(raw):
                    empty_row = True
                break

        # print_Grid(raw)

        # 3) If it was an empty row, delete it
        if empty_row and raw_exit_x in (0, len(raw)-1):
            del raw[raw_exit_x]
            # adjust raw_exit_x if it was the bottom row
            if raw_exit_x == len(raw):
                raw_exit_x = None  # will reset below
```

```
        # 4) Build sanitized grid (replace K with '.')
        sanitized = [ (r.replace('K','.') + '.'*M)[:M] for r in raw ]
        # print_Grid(sanitized)
        # 5) Compute padded exit_x,exit_y
        # after deletion, raw_exit_x may have changed if bottom
        if raw_exit_x is None:
            # bottom-edge exit: now at deleted-row index
            raw_exit_x = len(sanitized)
        # four edge cases:
        if raw_exit_x == 0 and empty_row:
            # top edge
            exit_x, exit_y = 0, raw_exit_y+1
        elif raw_exit_x == len(sanitized) and empty_row:
            # bottom edge
            exit_x, exit_y = raw_exit_x+1, raw_exit_y+1
        elif raw_exit_y == 0:
            # left edge
            exit_x, exit_y = raw_exit_x+1, 0
        elif raw_exit_y == M-1:
            # right edge
            exit_x, exit_y = raw_exit_x+1, raw_exit_y+1
        else:
            # interior K (shouldn't happen)
            exit_x = raw_exit_x+1
            exit_y = raw_exit_y+1

        # 6) Build the padded grid with sentinels
        H = len(sanitized)
        W = M
        padded = [['*']*(W+2) for _ in range(H+2)]
        for i in range(H):
            for j in range(W):
                padded[i+1][j+1] = sanitized[i][j]

        # 7) Overwrite the exit cell to '.' so we can slide through
        padded[exit_x][exit_y] = '.'

        # Save into self
        self.raw = [list(r) for r in sanitized]
        self.grid = padded
        self.rows, self.cols = H+2, W+2
        self.exit_pos = (exit_x, exit_y)

        # Run executable to get moves
        base_dir = os.path.dirname(__file__)
        exe_path = os.path.normpath(os.path.join(base_dir, '..', 'bin', 'gui.exe'))
        try:
            proc = subprocess.run([exe_path], input=input_data, capture_output=True,
text=True)
        except Exception as e:
            self.after(0, lambda: messagebox.showerror("Error", str(e)))
            self.after(0, lambda: self.solve_btn.config(state=tk.NORMAL))
            return

        # Parse moves from stdout
        self.moves = []
        nodes = None
        duration = None

        for line in proc.stdout.splitlines():
            if "Dikunjungi" in line and "ms" in line:
                match = re.search(r"Dikunjungi (\d+) simpul dalam ([\d.]+) ms", line)
                if match:
                    nodes = int(match.group(1))
                    duration = float(match.group(2))
            m = re.match(r"Move \d+: Piece (\w) moves (\w+) by (\d+)", line)
            if m:
                name, direction, dist = m.group(1), m.group(2), int(m.group(3))
                self.moves.append((name, direction, dist))

        self.nodes = nodes
        self.duration = duration

        # Draw and animate
```

```
        self.after(0, self.draw_background)
        self.after(0, self.draw_pieces)
        self.after(500, self.animate_next_move)
        status_msg = f"Status: {len(self.moves)} moves loaded"
        if nodes is not None and duration is not None:
            status_msg += f" | Simpul: {nodes} | Waktu: {duration} ms"

        self.after(0, lambda: self.status.config(text=status_msg))

    def draw_background(self):
        self.canvas.delete('all')
        w, h = self.cols*self.cell_size, self.rows*self.cell_size
        self.canvas.config(width=w, height=h)
        for i in range(self.rows):
            for j in range(self.cols):
                x0, y0 = j*self.cell_size, i*self.cell_size
                x1, y1 = x0+self.cell_size, y0+self.cell_size
                ch = self.grid[i][j]
                fill = 'darkgrey' if ch=='*' else 'lightgrey'
                self.canvas.create_rectangle(x0, y0, x1, y1, fill=fill, outline='black')

    def draw_pieces(self):
        for ids in self.piece_items.values():
            for _id in ids: self.canvas.delete(_id)
        self.piece_items.clear()
        for i in range(self.rows):
            for j in range(self.cols):
                ch = self.grid[i][j]
                if ch not in ('.','*',' '):
                    x0, y0 = j*self.cell_size, i*self.cell_size
                    x1, y1 = x0+self.cell_size, y0+self.cell_size
                    fill = 'red' if ch=='P' else 'skyblue'
                    rect = self.canvas.create_rectangle(x0, y0, x1, y1, fill=fill,
outline='black', tags=(ch,))
                    text = self.canvas.create_text((x0+x1)/2, (y0+y1)/2, text=ch,
font=('Arial',14,'bold'), tags=(ch,))
                    self.piece_items.setdefault(ch, []).extend([rect,text])

    def animate_next_move(self):
        if not self.moves:
            extra_info = ""
            if self.nodes is not None and self.duration is not None:
                extra_info = f" | Simpul: {self.nodes} | Waktu: {self.duration} ms"
            self.status.config(text=f"Animation complete{extra_info}")
            self.solve_btn.config(state=tk.NORMAL)
            return
        name, direction, dist = self.moves.pop(0)
        extra_info = ""
        if self.nodes is not None and self.duration is not None:
            extra_info = f" | Simpul: {self.nodes} | Waktu: {self.duration} ms"
        self.status.config(text=f"Moving {name} {direction} by {dist}{extra_info}")
        dx = (direction=='kanan') - (direction=='kiri')
        dy = (direction=='bawah') - (direction=='atas')
        dx *= self.cell_size; dy *= self.cell_size
        total_dx = dx//self.cell_size * dist
        total_dy = dy//self.cell_size * dist

        def step(k):
            if k==0:
                new = [row.copy() for row in self.grid]
                for i in range(self.rows):
                    for j in range(self.cols):
                        if self.grid[i][j]==name: new[i][j]='.'
                for i in range(self.rows):
                    for j in range(self.cols):
                        if self.grid[i][j]==name:
                            ni, nj = i+total_dy, j+total_dx
                            if 0<=ni<self.rows and 0<=nj<self.cols:
                                new[ni][nj]=name
                self.grid=new; self.draw_pieces()
                self.after(300, self.animate_next_move)
            else:
                for item in self.piece_items.get(name,[]):
                    self.canvas.move(item, dx, dy)
                self.after(150, lambda: step(k-1))
```

```
        step(dist)

if __name__=='__main__':
    RushHourGUI().mainloop()
```

BAB IV
PENGUJIAN DAN ANALISIS

4.1 Percobaan Program

Berikutnya, akan dilakukan demonstrasi dan percobaan untuk *interface* dari program.

4.1.1 Interface Masukan

Berikut beberapa demonstrasi masukan program.

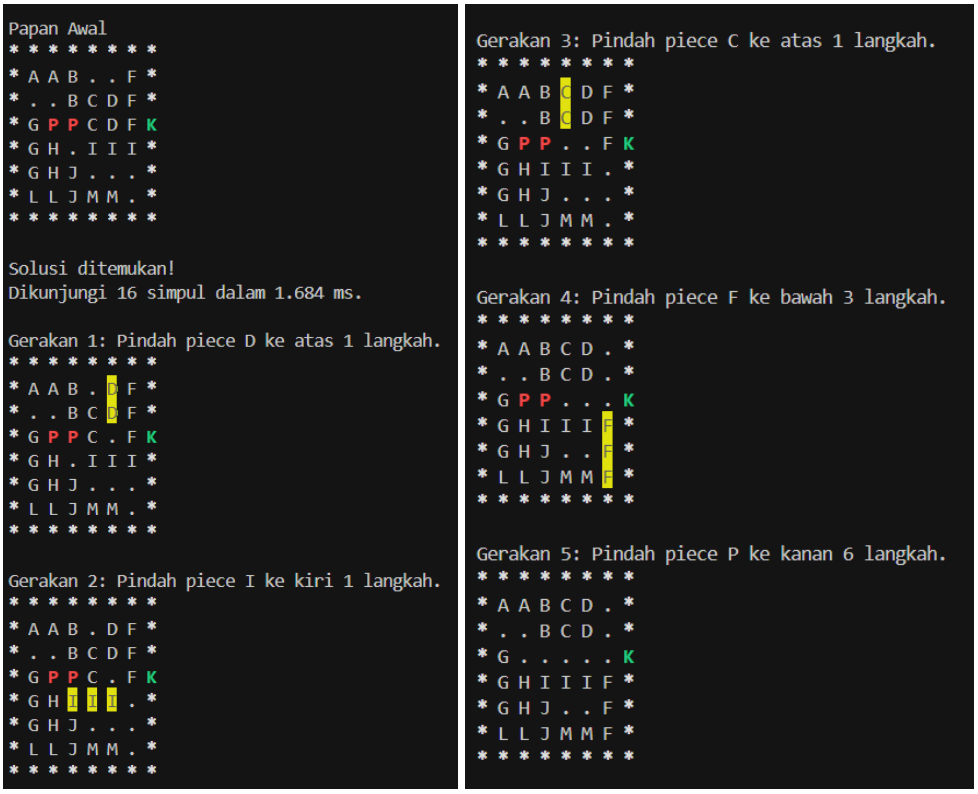
Landing terminal.

Masukan nama file.

Masukan pilihan algoritma.

4.1.2 Interface Keluaran

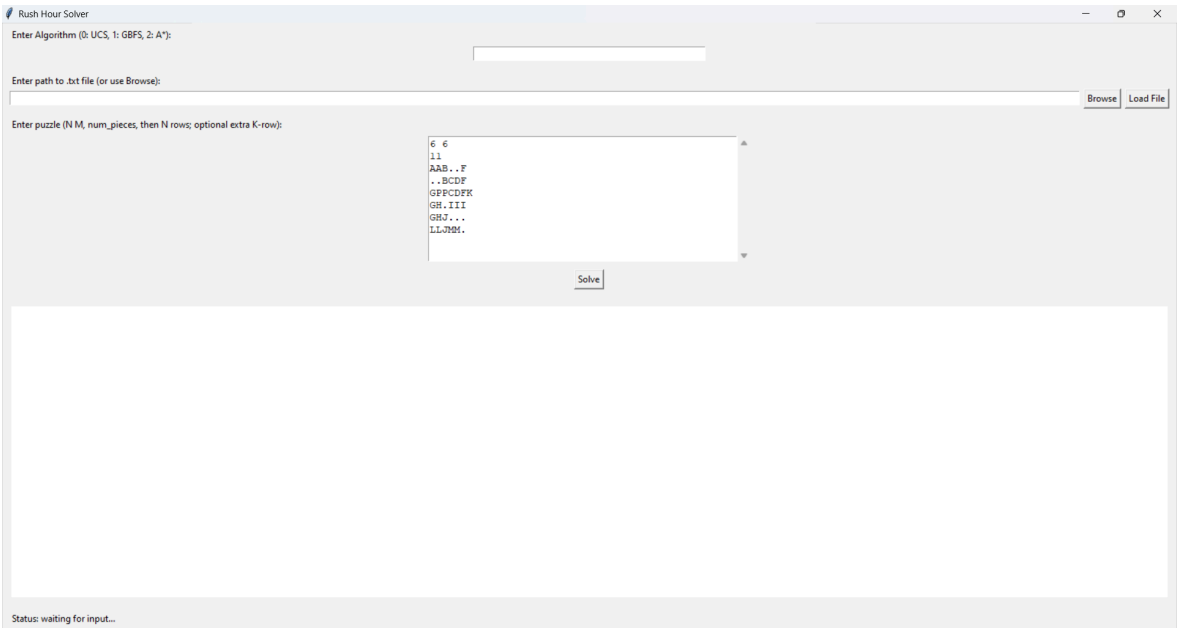
Berikut beberapa demonstrasi keluaran program.



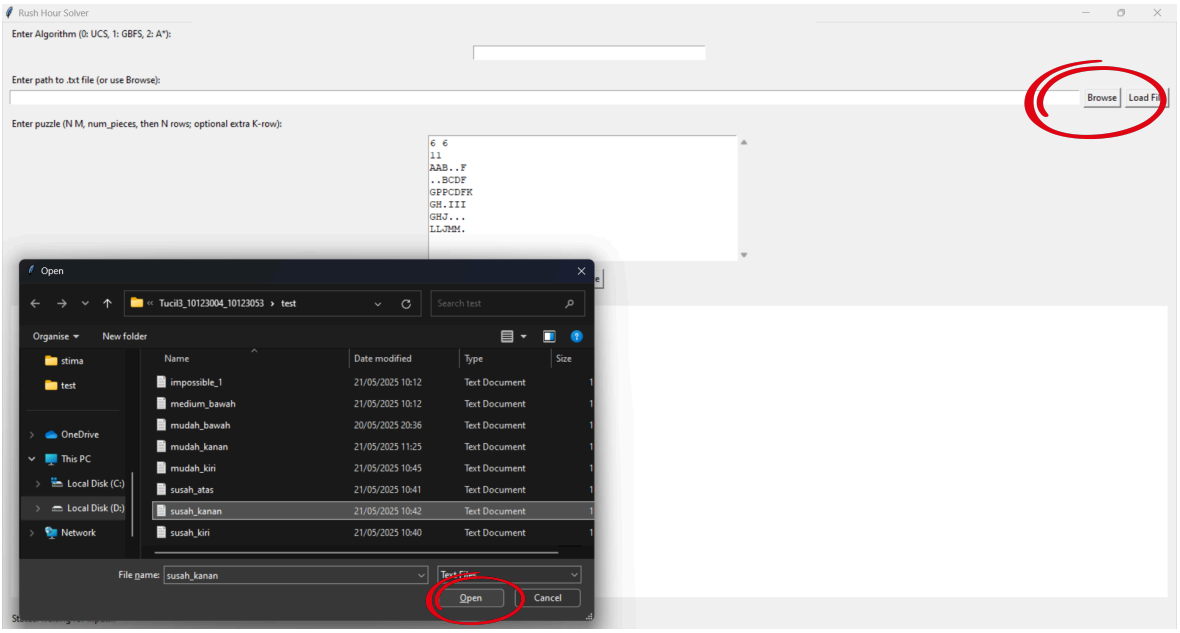
Keluaran lengkap hasil pencarian jalur

4.1.3. (*BONUS*) Preview GUI

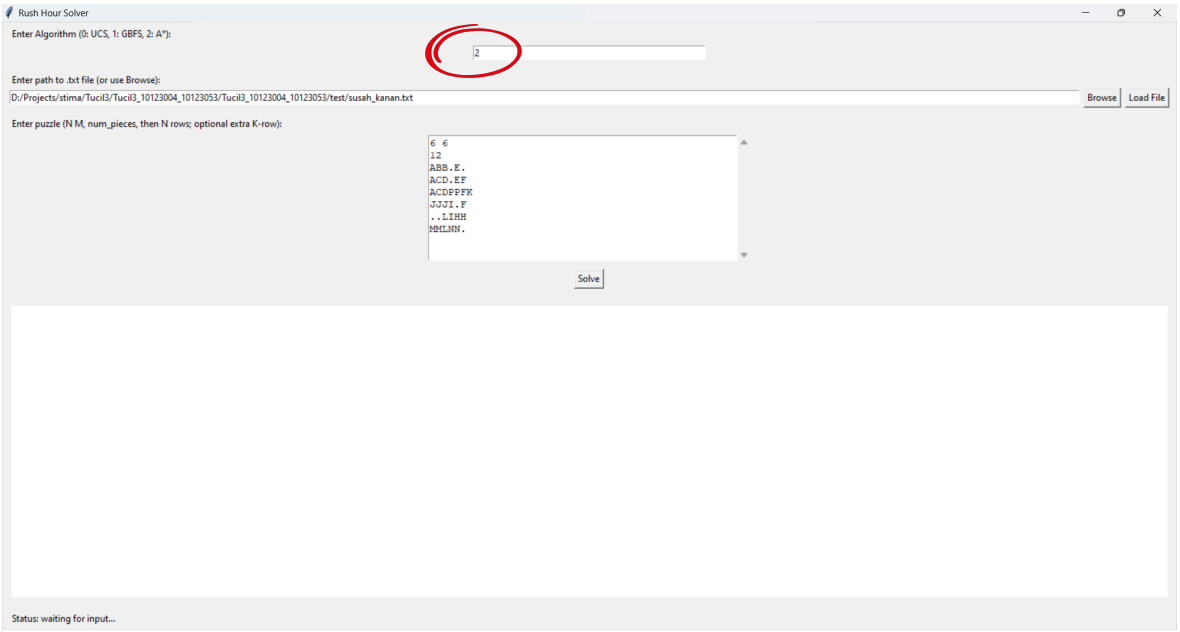
Berikut beberapa demonstrasi interface GUI yang dibuat.



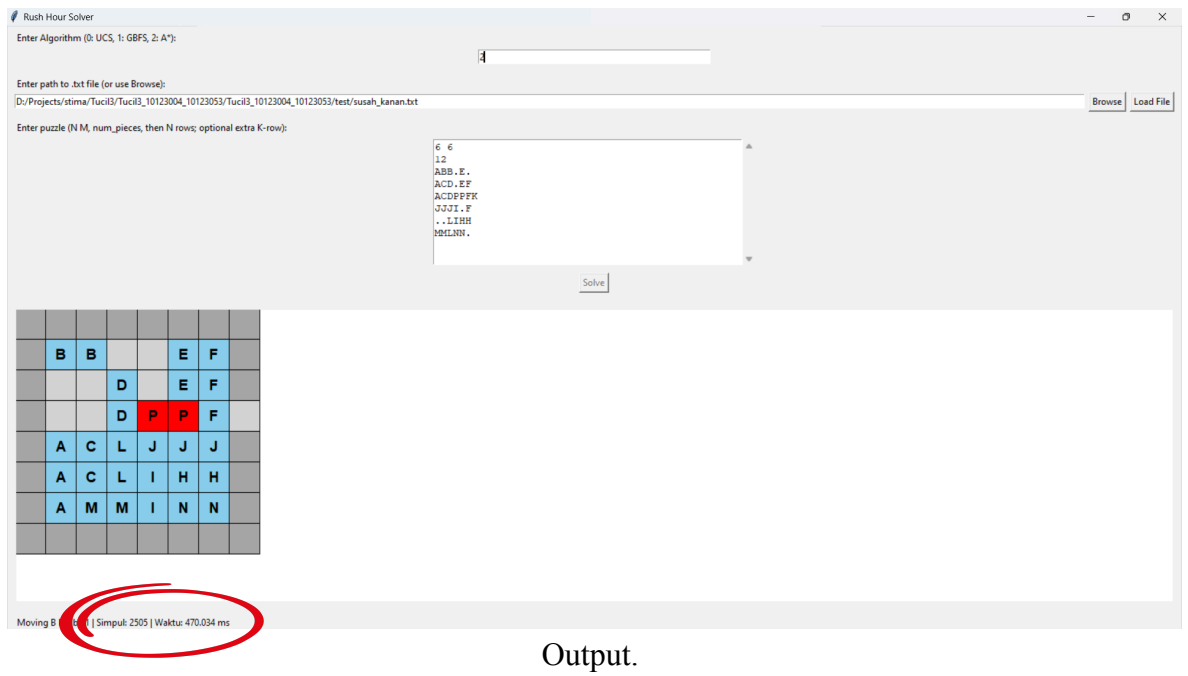
Landing GUI.



Masukan nama file.



Masukan pilihan algoritma.



Output.

4.2 Pengujian Algoritma *Pathfinding*

Berikutnya, akan diuji hasil solusi dan performa dari ketiga algoritma pencarian rute yang telah dibuat.

4.2.1 Pengujian UCS

Berikut pengujian untuk *Uniform Cost Search* (UCS).

```
Masukan file (.txt):
D:\Projects\stima\Tucil3\Tucil3_10123004_10123053\Tucil3_10123004_10123053\test\medium_b
awah.txt

6 6
9
AABB..
..C.DD
P.C.FF
PE..I.
.E..IG
HHHHIG
K


Berikut beberapa algoritma pathfinding.
  0. Uniform Cost Search (UCS)
  1. Greedy Best First Search
  2. A* Search
Pilih algoritma (0,1,2): 0

Papan Awal
* * * * * * * *
* A A B B . . *
* . . C . D D *
* P . C . F F *
* P E . . I . *
* . E . . I G *
* H H H H I G *
```

```
* K * * * * * *

Solusi ditemukan!
Dikunjungi 1090 simpul dalam 154.227 ms.

Gerakan 1: Pindah piece C ke bawah 2 langkah.
* * * * * * * *
* A A B B . . *
* . . . . D D *
* P . . . F F *
* P E C . I . *
* . E C . I G *
* H H H H I G *
* K * * * * * *

Gerakan 2: Pindah piece F ke kiri 2 langkah.
* * * * * * * *
* A A B B . . *
* . . . . D D *
* P . F F . . *
* P E C . I . *
* . E C . I G *
* H H H H I G *
* K * * * * * *

Gerakan 3: Pindah piece I ke atas 1 langkah.
* * * * * * * *
* A A B B . . *
* . . . . D D *
* P . F F I . *
* P E C . I . *
* . E C . I G *
* H H H H . G *
* K * * * * * *

Gerakan 4: Pindah piece H ke kanan 1 langkah.
* * * * * * * *
* A A B B . . *
* . . . . D D *
* P . F F I . *
* P E C . I . *
* . E C . I G *
* . H H H H G *
* K * * * * * *

Gerakan 5: Pindah piece P ke bawah 5 langkah.
* * * * * * * *
* A A B B . . *
* . . . . D D *
* . . F F I . *
* . E C . I . *
* . E C . I G *
* . H H H H G *
* K * * * * * *
```

4.2.2 Pengujian GBFS

Berikut pengujian untuk *Greedy Best-First Search* (GBFS).

```
Masukan file (.txt):
D:\Projects\stima\Tucil3\Tucil3_10123004_10123053\Tucil3_10123004_10123053\test\mudah_ka
nan.txt

6 6
11
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.
```

```
Berikut beberapa algoritma pathfinding.
  0. Uniform Cost Search (UCS)
  1. Greedy Best First Search
  2. A* Search
Pilih algoritma (0,1,2): 1

Papan Awal
* * * * * * * *
* A A B . . F *
* . . B C D F *
* G P P C D F K
* G H . I I I *
* G H J . . . *
* L L J M M . *
* * * * * * * *

Solusi ditemukan!
Dikunjungi 8 simpul dalam 1.010 ms.

Gerakan 1: Pindah piece C ke atas 1 langkah.
* * * * * * * *
* 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: Pindah piece D ke atas 1 langkah.
* * * * * * * *
* 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 3: Pindah piece I ke kiri 1 langkah.
* * * * * * * *
* 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: Pindah piece F ke bawah 3 langkah.
* * * * * * * *
* 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 5: Pindah piece P ke kanan 6 langkah.
* * * * * * * *
* 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 *
* * * * * * * *
```

```
Masukan file (.txt):
D:\Projects\stima\Tucil3\Tucil3_10123004_10123053\Tucil3_10123004_10123053\test\mudah_ki
ri.txt
```

```
6 6
4
.CC...
.B....
K.BAPP.
..A...
..A...
..DD..


Berikut beberapa algoritma pathfinding.
  0. Uniform Cost Search (UCS)
  1. Greedy Best First Search
  2. A* Search
Pilih algoritma (0,1,2): 1

Papan Awal
* * * * * * * *
* . C C . . . *
* . B . . . . *
K . B A P P . *
* . . A . . . *
* . . A . . . *
* . . D D . . *
* * * * * * * *

Solusi ditemukan!
Dikunjungi 9 simpul dalam 1.647 ms.

Gerakan 1: Pindah piece B ke bawah 3 langkah.
* * * * * * * *
* . C C . . . *
* . . . . . . *
K . . A P P . *
* . . A . . . *
* . B A . . . *
* . B D D . . *
* * * * * * * *

Gerakan 2: Pindah piece D ke kanan 1 langkah.
* * * * * * * *
* . C C . . . *
* . . . . . . *
K . . A P P . *
* . . A . . . *
* . B A . . . *
* . B . D D . *
* * * * * * * *

Gerakan 3: Pindah piece A ke bawah 1 langkah.
* * * * * * * *
* . C C . . . *
* . . . . . . *
K . . . P P . *
* . . A . . . *
* . B A . . . *
* . B A D D . *
* * * * * * * *

Gerakan 4: Pindah piece P ke kiri 6 langkah.
* * * * * * * *
* . C C . . . *
* . . . . . . *
K . . . . . . *
* . . A . . . *
* . B A . . . *
* . B A D D . *
* * * * * * * *
```

4.2.3 Pengujian A*

Berikut pengujian untuk *A* Search*.

```
Masukan file (.txt):
D:\Projects\stima\Tucil3\Tucil3_10123004_10123053\Tucil3_10123004_10123053\test\susah_ka
nan.txt

6 6
12
ABB.E.
ACD.EF
ACDPPFK
JJJI.F
..LIHH
MMLNN.


Berikut beberapa algoritma pathfinding.
  0. Uniform Cost Search (UCS)
  1. Greedy Best First Search
  2. A* Search
Pilih algoritma (0,1,2): 2

Papan Awal
* * * * * * * *
* A B B . E . *
* A C D . E F *
* A C D P P F K
* J J J I . F *
* . . L I H H *
* M M L N N . *
* * * * * * * *

Solusi ditemukan!
Dikunjungi 2505 simpul dalam 285.888 ms.

Gerakan 1: Pindah piece F ke atas 1 langkah.
* * * * * * * *
* A B B . E F *
* A C D . E F *
* A C D P P F K
* J J J I . . *
* . . L I H H *
* M M L N N . *
* * * * * * * *

Gerakan 2: Pindah piece N ke kanan 1 langkah.
* * * * * * * *
* A B B . E F *
* A C D . E F *
* A C D P P F K
* J J J I . . *
* . . L I H H *
* M M L . N N *
* * * * * * * *

Gerakan 3: Pindah piece I ke bawah 1 langkah.
* * * * * * * *
* A B B . E F *
* A C D . E F *
* A C D P P F K
* J J J . . . *
* . . L I H H *
* M M L I N N *
* * * * * * * *

Gerakan 4: Pindah piece J ke kanan 3 langkah.
* * * * * * * *
* A B B . E F *
* A C D . E F *
* A C D P P F K
* . . . J J J *
* . . L I H H *
```

```
* M M L I N N *
* * * * * * * *

Gerakan 5: Pindah piece L ke atas 1 langkah.
* * * * * * * *
* A B B . E F *
* A C D . E F *
* A C D P P F K
* . . L J J J *
* . . L I H H *
* M M . I N N *
* * * * * * * *

Gerakan 6: Pindah piece M ke kanan 1 langkah.
* * * * * * * *
* A B B . E F *
* A C D . E F *
* A C D P P F K
* . . L J J J *
* . . L I H H *
* . M M I N N *
* * * * * * * *

Gerakan 7: Pindah piece C ke bawah 2 langkah.
* * * * * * * *
* A B B . E F *
* A . D . E F *
* A . D P P F K
* . C L J J J *
* . C L I H H *
* . M M I N N *
* * * * * * * *

Gerakan 8: Pindah piece A ke bawah 3 langkah.
* * * * * * * *
* . B B . E F *
* . . D . E F *
* . . D P P F K
* A C L J J J *
* A C L I H H *
* A M M I N N *
* * * * * * * *

Gerakan 9: Pindah piece B ke kiri 1 langkah.
* * * * * * * *
* B B . . E F *
* . . D . E F *
* . . D P P F K
* A C L J J J *
* A C L I H H *
* A M M I N N *
* * * * * * * *

Gerakan 10: Pindah piece D ke atas 1 langkah.
* * * * * * * *
* B B D . E F *
* . . D . E F *
* . . . P P F K
* A C L J J J *
* A C L I H H *
* A M M I N N *
* * * * * * * *

Gerakan 11: Pindah piece P ke kiri 3 langkah.
* * * * * * * *
* B B D . E F *
* . . D . E F *
* P P . . . F K
* A C L J J J *
* A C L I H H *
* A M M I N N *
* * * * * * * *

Gerakan 12: Pindah piece E ke bawah 1 langkah.
* * * * * * * *
```

```
* B B D . . F *
* . . D . E F *
* P P . . E F K
* A C L J J J *
* A C L I H H *
* A M M I N N *
* * * * * * * *

Gerakan 13: Pindah piece D ke bawah 1 langkah.
* * * * * * * *
* B B . . . F *
* . . D . E F *
* P P D . E F K
* A C L J J J *
* A C L I H H *
* A M M I N N *
* * * * * * * *

Gerakan 14: Pindah piece B ke kanan 3 langkah.
* * * * * * * *
* . . . B B F *
* . . D . E F *
* P P D . E F K
* A C L J J J *
* A C L I H H *
* A M M I N N *
* * * * * * * *

Gerakan 15: Pindah piece D ke atas 1 langkah.
* * * * * * * *
* . . D B B F *
* . . D . E F *
* P P . . E F K
* A C L J J J *
* A C L I H H *
* A M M I N N *
* * * * * * * *

Gerakan 16: Pindah piece P ke kanan 2 langkah.
* * * * * * * *
* . . D B B F *
* . . D . E F *
* . . P P E F K
* A C L J J J *
* A C L I H H *
* A M M I N N *
* * * * * * * *

Gerakan 17: Pindah piece A ke atas 3 langkah.
* * * * * * * *
* A . D B B F *
* A . D . E F *
* A . P P E F K
* . C L J J J *
* . C L I H H *
* . M M I N N *
* * * * * * * *

Gerakan 18: Pindah piece M ke kiri 1 langkah.
* * * * * * * *
* A . D B B F *
* A . D . E F *
* A . P P E F K
* . C L J J J *
* . C L I H H *
* M M . I N N *
* * * * * * * *

Gerakan 19: Pindah piece C ke atas 3 langkah.
* * * * * * * *
* A C D B B F *
* A C D . E F *
* A . P P E F K
* . . L J J J *
* . . L I H H *
```

```
* M M . I N N *
* * * * * * * *

Gerakan 20: Pindah piece L ke bawah 1 langkah.
* * * * * * * *
* A C D B B F *
* A C D . E F *
* A . P P E F K
* . . . J J J *
* . . L I H H *
* M M L I N N *
* * * * * * * *

Gerakan 21: Pindah piece J ke kiri 3 langkah.
* * * * * * * *
* A C D B B F *
* A C D . E F *
* A . P P E F K
* J J J . . . *
* . . L I H H *
* M M L I N N *
* * * * * * * *

Gerakan 22: Pindah piece F ke bawah 1 langkah.
* * * * * * * *
* A C D B B . *
* A C D . E F *
* A . P P E F K
* J J J . . F *
* . . L I H H *
* M M L I N N *
* * * * * * * *

Gerakan 23: Pindah piece B ke kanan 1 langkah.
* * * * * * * *
* A C D . B B *
* A C D . E F *
* A . P P E F K
* J J J . . F *
* . . L I H H *
* M M L I N N *
* * * * * * * *

Gerakan 24: Pindah piece P ke kiri 1 langkah.
* * * * * * * *
* A C D . B B *
* A C D . E F *
* A P P . E F K
* J J J . . F *
* . . L I H H *
* M M L I N N *
* * * * * * * *

Gerakan 25: Pindah piece I ke atas 4 langkah.
* * * * * * * *
* A C D I B B *
* A C D I E F *
* A P P . E F K
* J J J . . F *
* . . L . H H *
* M M L . N N *
* * * * * * * *

Gerakan 26: Pindah piece J ke kanan 2 langkah.
* * * * * * * *
* A C D I B B *
* A C D I E F *
* A P P . E F K
* . . J J J F *
* . . L . H H *
* M M L . N N *
* * * * * * * *

Gerakan 27: Pindah piece P ke kanan 1 langkah.
* * * * * * * *
```

```
* A C D I B B *
* A C D I E F *
* A . P P E F K
* . . J J J F *
* . . L . H H *
* M M L . N N *
* * * * * * * *

Gerakan 28: Pindah piece C ke bawah 3 langkah.
* * * * * * * *
* A . D I B B *
* A . D I E F *
* A . P P E F K
* . C J J J F *
* . C L . H H *
* M M L . N N *
* * * * * * * *

Gerakan 29: Pindah piece P ke kiri 1 langkah.
* * * * * * * *
* A . D I B B *
* A . D I E F *
* A P P . E F K
* . C J J J F *
* . C L . H H *
* M M L . N N *
* * * * * * * *

Gerakan 30: Pindah piece I ke bawah 1 langkah.
* * * * * * * *
* A . D . B B *
* A . D I E F *
* A P P I E F K
* . C J J J F *
* . C L . H H *
* M M L . N N *
* * * * * * * *

Gerakan 31: Pindah piece B ke kiri 1 langkah.
* * * * * * * *
* A . D B B . *
* A . D I E F *
* A P P I E F K
* . C J J J F *
* . C L . H H *
* M M L . N N *
* * * * * * * *

Gerakan 32: Pindah piece F ke atas 1 langkah.
* * * * * * * *
* A . D B B F *
* A . D I E F *
* A P P I E F K
* . C J J J . *
* . C L . H H *
* M M L . N N *
* * * * * * * *

Gerakan 33: Pindah piece J ke kanan 1 langkah.
* * * * * * * *
* A . D B B F *
* A . D I E F *
* A P P I E F K
* . C . J J J *
* . C L . H H *
* M M L . N N *
* * * * * * * *

Gerakan 34: Pindah piece L ke atas 1 langkah.
* * * * * * * *
* A . D B B F *
* A . D I E F *
* A P P I E F K
* . C L J J J *
* . C L . H H *
```

```
* M M . . N N *
* * * * * * * *

Gerakan 35: Pindah piece M ke kanan 1 langkah.
* * * * * * * *
* A . D B B F *
* A . D I E F *
* A P P I E F K
* . C L J J J *
* . C L . H H *
* . M M . N N *
* * * * * * * *

Gerakan 36: Pindah piece A ke bawah 3 langkah.
* * * * * * * *
* . . D B B F *
* . . D I E F *
* . P P I E F K
* A C L J J J *
* A C L . H H *
* A M M . N N *
* * * * * * * *

Gerakan 37: Pindah piece P ke kiri 1 langkah.
* * * * * * * *
* . . D B B F *
* . . D I E F *
* P P . I E F K
* A C L J J J *
* A C L . H H *
* A M M . N N *
* * * * * * * *

Gerakan 38: Pindah piece D ke bawah 1 langkah.
* * * * * * * *
* . . . B B F *
* . . D I E F *
* P P D I E F K
* A C L J J J *
* A C L . H H *
* A M M . N N *
* * * * * * * *

Gerakan 39: Pindah piece B ke kiri 3 langkah.
* * * * * * * *
* B B . . . F *
* . . D I E F *
* P P D I E F K
* A C L J J J *
* A C L . H H *
* A M M . N N *
* * * * * * * *

Gerakan 40: Pindah piece N ke kiri 1 langkah.
* * * * * * * *
* B B . . . F *
* . . D I E F *
* P P D I E F K
* A C L J J J *
* A C L . H H *
* A M M N N . *
* * * * * * * *

Gerakan 41: Pindah piece D ke atas 1 langkah.
* * * * * * * *
* B B D . . F *
* . . D I E F *
* P P . I E F K
* A C L J J J *
* A C L . H H *
* A M M N N . *
* * * * * * * *

Gerakan 42: Pindah piece P ke kanan 1 langkah.
* * * * * * * *
```

```
* B B D . . F *
* . . D I E F *
* . P P I E F K
* A C L J J J *
* A C L . H H *
* A M M N N . *
* * * * * * * *

Gerakan 43: Pindah piece A ke atas 1 langkah.
* * * * * * * *
* B B D . . F *
* . . D I E F *
* A P P I E F K
* A C L J J J *
* A C L . H H *
* . M M N N . *
* * * * * * * *

Gerakan 44: Pindah piece M ke kiri 1 langkah.
* * * * * * * *
* B B D . . F *
* . . D I E F *
* A P P I E F K
* A C L J J J *
* A C L . H H *
* M M . N N . *
* * * * * * * *

Gerakan 45: Pindah piece E ke atas 1 langkah.
* * * * * * * *
* B B D . E F *
* . . D I E F *
* A P P I . F K
* A C L J J J *
* A C L . H H *
* M M . N N . *
* * * * * * * *

Gerakan 46: Pindah piece L ke bawah 1 langkah.
* * * * * * * *
* B B D . E F *
* . . D I E F *
* A P P I . F K
* A C . J J J *
* A C L . H H *
* M M L N N . *
* * * * * * * *

Gerakan 47: Pindah piece H ke kiri 1 langkah.
* * * * * * * *
* B B D . E F *
* . . D I E F *
* A P P I . F K
* A C . J J J *
* A C L H H . *
* M M L N N . *
* * * * * * * *

Gerakan 48: Pindah piece J ke kiri 1 langkah.
* * * * * * * *
* B B D . E F *
* . . D I E F *
* A P P I . F K
* A C J J J . *
* A C L H H . *
* M M L N N . *
* * * * * * * *

Gerakan 49: Pindah piece F ke bawah 3 langkah.
* * * * * * * *
* B B D . E . *
* . . D I E . *
* A P P I . . K
* A C J J J F *
* A C L H H F *
```

```
* M M L N N F *
* * * * * * * *

Gerakan 50: Pindah piece I ke atas 1 langkah.
* * * * * * * *
* B B D I E . *
* . . D I E . *
* A P P . . . K
* A C J J J F *
* A C L H H F *
* M M L N N F *
* * * * * * * *

Gerakan 51: Pindah piece P ke kanan 6 langkah.
* * * * * * * *
* B B D I E . *
* . . D I E . *
* A . . . . . K
* A C J J J F *
* A C L H H F *
* M M L N N F *
* * * * * * * *
```

```
Masukan file (.txt):
D:\Projects\stima\Tucil3\Tucil3_10123004_10123053\Tucil3_10123004_10123053\test\susah_at
as.txt

Berikut beberapa algoritma pathfinding.
  0. Uniform Cost Search (UCS)
  1. Greedy Best First Search
  2. A* Search
Pilih algoritma (0,1,2): 2

Papan Awal
* * * K * * * *
* A B C C E E *
* A B D D F G *
* . B P I F G *
* J J P I F H *
* Z . M M M H *
* Z L L N N N *
* * * * * * * *

Solusi ditemukan!
Dikunjungi 56 simpul dalam 2.993 ms.

Gerakan 1: Pindah piece M ke kiri 1 langkah.
* * * K * * * *
* A B C C E E *
* A B D D F G *
* . B P I F G *
* J J P I F H *
* Z M M M . H *
* Z L L N N N *
* * * * * * * *

Gerakan 2: Pindah piece F ke bawah 1 langkah.
* * * K * * * *
* A B C C E E *
* A B D D . G *
* . B P I F G *
* J J P I F H *
* Z M M M F H *
* Z L L N N N *
* * * * * * * *

Gerakan 3: Pindah piece D ke kanan 1 langkah.
* * * K * * * *
* A B C C E E *
* A B . D D G *
* . B P I F G *
* J J P I F H *
* Z M M M F H *
* Z L L N N N *
```

```
* * * * * * * * *

Gerakan 4: Pindah piece P ke atas 1 langkah.
* * * K * * * *
* A B C C E E *
* A B P D D G *
* . B P I F G *
* J J . I F H *
* Z M M M F H *
* Z L L N N N *
* * * * * * * * *

Gerakan 5: Pindah piece J ke kanan 1 langkah.
* * * K * * * *
* A B C C E E *
* A B P D D G *
* . B P I F G *
* . J J I F H *
* Z M M M F H *
* Z L L N N N *
* * * * * * * * *

Gerakan 6: Pindah piece Z ke atas 1 langkah.
* * * K * * * *
* A B C C E E *
* A B P D D G *
* . B P I F G *
* Z J J I F H *
* Z M M M F H *
* . L L N N N *
* * * * * * * * *

Gerakan 7: Pindah piece L ke kiri 1 langkah.
* * * K * * * *
* A B C C E E *
* A B P D D G *
* . B P I F G *
* Z J J I F H *
* Z M M M F H *
* L L . N N N *
* * * * * * * * *

Gerakan 8: Pindah piece N ke kiri 1 langkah.
* * * K * * * *
* A B C C E E *
* A B P D D G *
* . B P I F G *
* Z J J I F H *
* Z M M M F H *
* L L N N N . *
* * * * * * * * *

Gerakan 9: Pindah piece H ke bawah 1 langkah.
* * * K * * * *
* A B C C E E *
* A B P D D G *
* . B P I F G *
* Z J J I F . *
* Z M M M F H *
* L L N N N H *
* * * * * * * * *

Gerakan 10: Pindah piece G ke bawah 1 langkah.
* * * K * * * *
* A B C C E E *
* A B P D D . *
* . B P I F G *
* Z J J I F G *
* Z M M M F H *
* L L N N N H *
* * * * * * * * *

Gerakan 11: Pindah piece D ke kanan 1 langkah.
* * * K * * * *
* A B C C E E *
```

```
* A B P . D D *
* . B P I F G *
* Z J J I F G *
* Z M M M F H *
* L L N N N H *
* * * * * * * *

Gerakan 12: Pindah piece I ke atas 1 langkah.
* * * K * * * *
* A B C C E E *
* A B P I D D *
* . B P I F G *
* Z J J . F G *
* Z M M M F H *
* L L N N N H *
* * * * * * * *

Gerakan 13: Pindah piece J ke kanan 1 langkah.
* * * K * * * *
* A B C C E E *
* A B P I D D *
* . B P I F G *
* Z . J J F G *
* Z M M M F H *
* L L N N N H *
* * * * * * * *

Gerakan 14: Pindah piece B ke bawah 1 langkah.
* * * K * * * *
* A . C C E E *
* A B P I D D *
* . B P I F G *
* Z B J J F G *
* Z M M M F H *
* L L N N N H *
* * * * * * * *

Gerakan 15: Pindah piece A ke bawah 1 langkah.
* * * K * * * *
* . . C C E E *
* A B P I D D *
* A B P I F G *
* Z B J J F G *
* Z M M M F H *
* L L N N N H *
* * * * * * * *

Gerakan 16: Pindah piece C ke kiri 2 langkah.
* * * K * * * *
* C C . . E E *
* A B P I D D *
* A B P I F G *
* Z B J J F G *
* Z M M M F H *
* L L N N N H *
* * * * * * * *

Gerakan 17: Pindah piece P ke atas 4 langkah.
* * * K * * * *
* C C . . E E *
* A B . I D D *
* A B . I F G *
* Z B J J F G *
* Z M M M F H *
* L L N N N H *
* * * * * * * *
```

4.3 Analisis Algoritma

Berdasarkan teori dan hasil pengujian yang dilakukan, akan dilakukan perbandingan dari ketiga algoritma.

4.3.1. Perbandingan Empiris

Untuk membandingkan performa dari masing-masing algoritma, kami mengujikan ketiga algoritma dengan test case yang sama dimana test case dirasa cukup berat untuk dapet meng-stress test algoritma. Berikut rangkuman dari hasil percobaan di atas untuk merepresentasikan performa dari setiap algoritma pencarian rute.

Test Case	UCS	GBFS	A*
`susah_atas.txt`	3.555 ms	2.472 ms	2.905 ms
	58 simpul	52 simpul	56 simpul
	17 Gerakan	18 gerakan	17 gerakan
`susah_kanan.txt`	325.732 ms	245.186 ms	280.941 ms
	2823 simpul	1733 simpul	2505 simpul
	51 gerakan	136 gerakan	51 gerakan
`susah_kiri.txt`	336.317 ms	491.387 ms	285.244 ms
	2480 simpul	1732 simpul	2517 simpul
	51 gerakan	136 gerakan	51 gerakan

Jika membandingkan dari waktu runtime, algoritma UCS selalu lebih lambat dari BGFS dan A* karena sifat dari UCS yang sifatnya yang eksploratif dan menelusuri semua simpul, namun tetap mendapatkan solusi optimal. GBFS umumnya lebih cepat, tetapi tidak konsisten (pada kasus `susah_kiri.txt` paling lambat dengan 491.387 ms) karena hanya mempertimbangkan heuristik $g(n)$ tanpa mempertimbangkan biaya aktual $f(n)$ sehingga dapat terjebak dalam mengeksplorasi simpul yang tidak mungkin jadi jawaban. Pencarian A* lebih cepat dari UCS tetapi lebih lambat dari GBFS di beberapa kasus, kendati demikian A* menghasilkan solusi optimal dibandingkan GBFS, tetapi lebih efisien daripada UCS.

Dalam hal jumlah simpul yang dieksplorasi GBFS secara umum mengunjungi simpul yang lebih sedikit dibandingkan UCS dan A*, karena heuristic yang selalu mengarahkan pada solusi. Algoritma A* mengunjungi simpul yang lebih banyak dari GBFS tetapi lebih sedikit

dari UCS karena kombinasi $f(n)$ dan $g(n)$ membuat A* mencari solusi optimal tapi tetap terarah. UCS mengunjungi simpul paling banyak karena merupakan *uninformed search* dan mengunjungi semua simpul yang dapat menjadi solusi.

Dari faktor optimalitas solusi, UCS menghasilkan solusi optimal karena tetap menggunakan biaya aktual. Algoritma A* menghasilkan solusi optimal karena menggunakan biaya aktual dibantu dengan heuristic yang *admissible* dan *consistent*. Sedangkan GBFS tidak menghasilkan solusi optimal karena sifat greedy yang dapat membuat terjebak di local optima.

4.3.2. Kompleksitas Teoritis

Selebihnya, kami akan menganalisis performa teoritis dari ketiga algoritma pencarian rute secara lebih kuantitatif menggunakan notasi Big O. Untuk membantu, ada beberapa variabel yang akan digunakan.

1. b: faktor percabangan, yaitu rata-rata gerakan kendaraan yang mungkin dari suatu konfigurasi, atau jumlah anak status yang mungkin dari suatu status
2. d: kedalaman solusi, yaitu jumlah gerakan yang diperlukan untuk mencapai solusi optimal dari status awal
3. m: kedalaman pencarian maksimum, yaitu jumlah gerakan maksimum untuk mencapai suatu solusi, sehingga $m > d$
4. l: dimensi papan (ambil saja maksimum dari panjang dan lebar)
5. S: jumlah status atau konfigurasi unik yang dapat dicapai dari status awal, yang untuk suatu papan Rush Hour sederhana seharusnya berhingga

Pertama, sesuai rekayasa yang dilakukan di Bab 2, karena harga jarak suatu simpul status hanyalah kedalamannya di pohon pencarian, UCS bekerja seperti BFS untuk persoalan ini. Maka, UCS akan mengunjungi setiap status secara bertahap, sehingga kompleksitas waktu UCS berorde $O(b^d)$. Selebihnya, semua simpul yang dikunjungi disimpan dalam antrian prioritas pada setiap level, sehingga kompleksitas ruangnya juga $O(b^d)$. Karena jumlah konfigurasi papan juga berhingga, kita bisa menulis kedua kompleksitas ini juga sebagai $O(S)$ untuk kasus paling buruknya, karena UCS tidak akan melewati status (hanya berhenti jika sudah mencapai solusi).

Untuk GBFS, efisiensi akan bergantung heuristik yang digunakan. Saat heuristik sempurna (yaitu $h(n) = h'(n)$ sehingga sebenarnya sudah diketahui solusi optimalnya), kompleksitas waktu maupun ruang dari GBFS hanyalah $O(d)$. Di kasus terburuk, heuristik akan keliru, sehingga suatu solusi bisa saja tidak akan ditemukan walaupun pencarian sangat dalam, sehingga dapat diperoleh kompleksitas berorde $O(b^m)$. Di atas kompleksitas algoritma ini, ada juga kompleksitas untuk menghitung heuristiknya sendiri. Dalam tugas ini, diterapkan heuristik jumlah kendaraan yang memblokir jalur keluar. Ini heuristik yang *admissible*, sehingga dia tidak keliru sekali, tetapi tidak selalu konsisten. Maka, kasus terburuknya tetap $O(b^m)$, tergantung status awal. Adapun tapi bahwa perhitungan heuristik ini tidak terlalu sulit, dan hanya menjelajah maksimal seluruh sel pada suatu baris atau kolom, sehingga dia hanya berorde $O(l)$.

Akhirnya, algoritma A* menggabungkan kedua konsep UCS dan GBFS dengan menggunakan jumlah kedua fungsi evaluasi. Ini menghasilkan batas atas dan bawah yang lebih baik untuk A*. Di kasus terburuk, misalnya heuristik gagal total atau bahkan heuristik nol, A* menjadi UCS, sehingga kompleksitas waktu batas atas menjadi $O(b^d)$ atau $O(S)$. Di kasus terbaik, heuristik langsung mengarah pada solusi optimal, sehingga A* menjadi lebih mirip GBFS, maka kompleksitas waktu batas bawah menjadi $O(d)$. Fakta yang serupa dapat dikatakan untuk kompleksitas ruang. Karena batas yang lebih ketat ini, seharusnya A* adalah algoritma yang lebih *well-rounded* untuk permasalah Rush Hour. Lalu, karena heuristik bersifat *admissible*, A* pasti akan menghasilkan solusi optimal. Ini menghasilkan kompleksitas yang berada di tengah-tengah antara $O(d)$ dan $O(b^d)$.

DAFTAR PUSTAKA

Munir, Rinaldi. "Penentuan rute (Route/Path Planning) - Bagian 1." *Informatika STEI ITB*. Accessed 20 Mei, 2025. Available at: informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf.

Munir, Rinaldi. "Penentuan rute (Route/Path Planning) - Bagian 2." *Informatika STEI ITB*. Accessed 20 Mei, 2025. Available at: informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf.

LAMPIRAN

Berikut lampiran *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.	✔	

Berikut lampiran tautan.

Repository GitHub	github.com/adielrum/Tucil3_10123004_10123053