# Laporan Tugas Kecil 3

Disusun untuk memenuhi tugas kecil 3 mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2024/2025


**Oleh**

**Stefan Mattew Susanto 13523020**

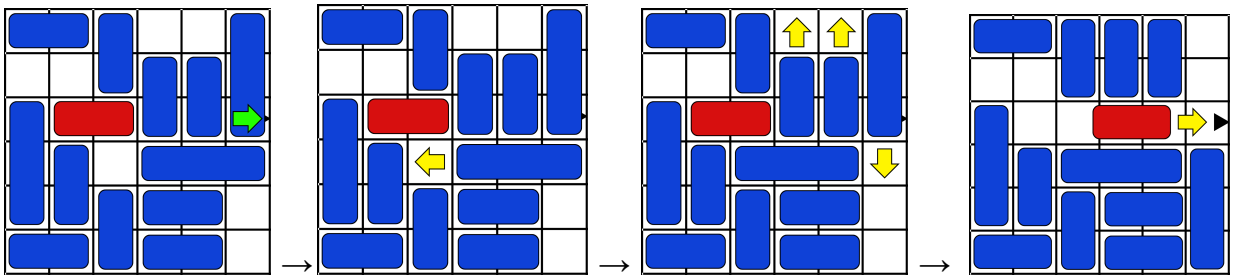
**PROGRAM STUDI TEKNIK INFORMATIKA**
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**
**INSTITUT TEKNOLOGI BANDUNG**
**BANDUNG**
**2025**

# Daftar Isi

# Deskripsi Masalah


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.

Rush Hour memiliki beberapa komponen penting, seperti papan, piece, primary piece(‘P’) , pintu keluar (‘K’), gerakan. Ilustrasi kasus sebagai berikut:



# Analisis Algoritma

## 2.1. Algoritma yang digunakan

- Algoritma UCS

  UCS adalah algoritma pathfinding yang menggunakan cost dari start sebagai dasar untuk eksplorasi. Algoritma ini menghitung cost akumulatif terkecil dari start node ke node n untuk selanjutnya dapat memilih jalur yang paling efisien. Algoritma UCS dilambangkan sebagai g(n). UCS tidak menggunakan fungsi heuristic dalam pemilihan jalur. Sehingga fungsi UCS didefinisikan sebagai f(n) = g(n).

  Langkah - langkah UCS :

  - Algoritma UCS dimulai dari start node.
  - Node akan dimasukkan ke dalam priorityqueue dan memasukkan nilai 0 sebagai state awal.
  - Algoritma akan melakukan eksplorasi node dan memasukkan nilai state ke dalam priority queue.
  - Algoritma akan menuju node dengan nilai state (g) terkecil dengan mengambil nilai dari priority queue.
  - Algoritma akan terus menerus melakukan penelusuran hingga mencapai node goal. Algoritma melakukan generate child nodes untuk melakukan eksplorasi an mengupdate state baru yang akan dimasukkan ke dalam priority queue.
  - Jika goal telah dicapai, algoritma akan berhenti dan mengembalikan jalur.

- Algoritma Greedy Best First Search

  Greedy Best First Search adalah algoritma pathfinding yang memilih jalur yang diperkirakan memiliki cost terendah untuk mencapai target/goal. Algotima ini menggunakan nilai dari fungsi heuristik (h) terkecil sebagai cara untuk pemilihan jalur. Algoritma ini tidak mempertimbangkan cost yang digunakan dari start ke node n. Sehingga fungsi GBFS didefinisikan sebagai f(n) = h(n).

  Langkah-langkah GBFS:

  - Algoritma akan dimulai dari start node.
  - Start node akan dimasukkan ke dalam priority queue dengan nilai nol.
  - Algoritma akan memulai eksplorasi dengan menghitung nilai heuristik dari langkah yang dapat diambil
  - Nilai state akan dimasukkan ke dalam priority queue
  - Algoritma akan menuju node dengan menggunakan jalur dengan nilai h terkecil dalam priority queue.
  - Eksplorasi node akan dilanjutkan dari node terbaru dengan menggunakan generate child nodes hingga mencapai node goal.

- Saat goal telah dicapai, maka algoritma akan berhenti dan mengembalikan jalur yang digunakan dari start node ke goal node.

Heuristik merupakan suatu metode untuk menghitung nilai dengan pendekatan tertentu supaya dapat menghasilkan jalur yang paling efisien. Heuristik yang digunakan pada program ini adalah heuristik manhatan Distance. Heuristic ini mengukur jarak antara posisi mobil/piece ('P') ke exit ('K'). Jarak akan diukur tanpa mempertimbangkan jumlah mobil/piece yang menghalangi jalur keluar.

- Algoritma A*
  Algoritma A star/ A* adalah algoritma patfinding yang menggabungkan GBFS dan UCS untuk mendapatkan jalur dengan cost terkecil. A* membutuhkan fungsi g dan fungsi h sehingga fungsi A* adalah f(n) = g(n) + h(n). A* akan mencari jalur yang mempertimbangkan cost dari start node ke node n dan mempertimbangkan fungsi heuristik dengan aspek jarak yang perlu ditempuh ke goal node.
  Langkah- langkah A*:
  - Algoritma akan dimulai dari start node yakni kondisi pertama board.
  - Start node akan dimasukkan ke dalam priority queue dengan nilai g dan h nol.
  - Algoritma akan melakukan eksplorasi dengan menghitung jalur heuristik dan cost dari start.
  - Nilai state berupa g dan h akan dijumlahkan dan dimasukkan ke dalam variabel f pada state dan akan dimasukkan ke dalam priority queue.
  - Algoritma akan melanjutkan dengan node pertama pada priority queue dimana nilai f yang paling kecil.
  - Penjelajahan node akan dilakukan kembali dengan menggunakan fungsi generate child nodes dan akan kembali dimasukkan.
  - Jika goal node telah dicapai maka path akan dikembalikan dan akan mendapatkan jalur yang paling efisien karena menggunakan dua pendekatan sekaligus.

## 2.2. Analisis Algoritma

Pada algoritma UCS dan A* menggunakan fungsi f(n) dan g(n). g(n) merupakan cost yang diperlukan dari start ke node n. Nilai g(n) memperhitungkan jarak dari awal hingga node saat ini. Sedangkan f(n) merupakan fungsi yang memperhitungkan total biaya yang diperlukan dari start node hingga sampai node terakhir atau node goal dengan melewati node n dan pertimbangan tertentu. Pada UCS f(n) akan sama dengan nilai g(n) karena UCS tidak memperhitungkan heuristic sebagai aspek di dalamnya. Sedangkan A*, f(n) = g(n) +h(n) karena A* menggunakan estimasi jarak node n ke goal. f(n) akan mengestimasi cost yang paling murah berdasarkan jarak yang paling pendek dari start dan jarak yang paling pendek ke goal.

Heuristik akan disebut admissible jika untuk setiap node n, h(n) <= h*(n) dimana h*(n) adalah cost sebenarnya untuk mencapai goal state dari n. Admissible heuristic tidak melebih-lebihkan cost untuk mencapai goal dan bersifat optimistic. Heuristic manhattan distance pada A* bersifat admisible karena heuristic ini mengukur jarak antara posisi kendaraan/piece dengan exit dengan menghitung jumlah step. Jarak pada heuristik akan selalu lebih kecil dari ukuran grid tanpa melebih-lebihkan dari tujuan. Jika heuristic berdasarkan jumlah mobil yang menhalangi tidak bersifat admisible karena terdapat peluang bahwa heuristik akan melebih lebihkan cost untuk mencapai tujuan dari yang sebenarnya.

Algoritma UCS berbeda dengan algoritma BFS karena algoritma UCS berfokus pada cost yang diperlukan tanpa melihat atau tidak berdasarkan depth suatu node. Pada BFS, node akan dieksplor bertahap sesuai kedalaman. BFS lebih sering diterapkan dalam uninformed search sehingga cost bersifat tidak diperhitungkan atau diabaikan. UCS pada game ini diterapkan dengan melihat nilai cost dari start hingga node n dan akan mengambil jalur berdasarkan nilai g terkecil. Node akan dipilih berdasarkan urutan pada priority queue dengan nilai g terkecil. Jika BFS setiap node / gerakan piece akan dieksplor semua lalu tingkatan selanjutnya akan dicek juga meskipun itu bukan jalur yang paling pendek. Sehingga UCS akan lebih efisien dan lebih cocok digunakan pada game rush hour ini.

Secara teori algoritma A* akan lebih efisien dibanding algotima UCS pada penyelesaian game rush hour karena A* akan mempertimbangkan aspek yang lebih banyak, yakni kondisi saat ini dan estimasi kondisi yang akan datang (heuristic). Dengan heuristic A*, jalur yang akan diambil akan lebih mungkin untuk lebih singkat. Jalur A* memungkinkan untuk menghindari jalur jalur yang diprediksi membuat goal node lebih jauh. Sedangkan UCS akan terus mengeskplor jalur tanpa mempertimbangkan jalur yang diambil dan dapat lebih mungkin untuk menempuh jalur yang tidak perlu dicek.

Secara teoritis, algortima Greedy Best First Search tidak menjamin solusi yang optimal karena algoritma ini tidak memperhitungkan cost yang telah ditempuh dari start hingga current node. Algoritma ini hanya mempertimbangkan heuristic sebagai suatu estimasi nilai yang memungkinkan untuk mencapai lebih dekat tanpa mengetahui sebenarnya nilai yang sudah ditempuh. GBFS dapat menempuh jalur yang bukan paling singkat untuk mencapai goalnode.

Algortima GBFS, UCS dan A* memiliki karakteristik dan pertimbangan masing masing. GBFS akan mempertimbangkan nilai heuristic yang mencari estimasi cost untuk sampai goal node. GBFS mungkin tidak menemukan jalur yang paling optimal karena tidak menghitung jarak yang telah ditempuh. Namun secara teori, proses GBFS cenderung lebih singkat dibanding UCS dan A*. UCS akan menempuh node dengan nilai cost yang terkecil dari startnode hingga current

node. UCS akan menempuh jalur yang paling singkat atau cost terendah dari start node ke goal node namun kurang optimal dalam segi waktu. Algoritma A* mempertimbangkan nilai heuristic dan nilai cost yang telah ditempuh. Sehingga A* memungkinkan untuk mendapat jalur yang paling efisien dengan waktu yang lebih singkat dibanding UCS.

# Implementasi Program

## 3.1. Struktur Data

Program ini memiliki struktur sebagai berikut

```
└── Tucil3_13523020
    ├── src
    │   ├── board.h
    │   ├── board.c
    │   ├── piece.h
    │   ├── piece.c
    │   ├── priorityqueue.h
    │   ├── priorityqueue.c
    │   ├── pathfinder.h
    │   ├── pathfinder.c
    │   ├── hashset.h
    │   ├── hashset.c
    │   ├── utils.h
    │   ├── utils.c
    │   └── main.c
    ├── doc
    │   └── Tucil3_13523020
    ├── bin
    │   └── main.exe
    ├── test
    │   ├── tester1.txt
    │   ├── tester2.txt
    │   ├── tester3.txt
    │   └── tester4.txt
    └── README.md
```

## 3.2. Source Program

Program menggunakan bahasa C secara keseluruhan. Fungsi yang digunakan terdapat pada header sebagai berikut:

- board.h

```c
#ifndef BOARD_H
#define BOARD_H
```

```c
#include "piece.h"


typedef struct {
    char pieceId;
    char direction; // 'U', 'D', 'L', 'R'
    int steps;
} Move;

typedef struct {
    int rows;
    int cols;
    char** grid;
    int numPieces;
    Piece *pieces;
    char primaryPieceID;
    int primaryIndex;
    int exit_row;
    int exit_col;
    Move *solution;
    int solutionLength;
} Board;

extern int totalCheck;
void initBoard(Board *board);
void printBoard(const Board *board);

bool compareBoards(const Board *a, const Board *b);

int findPieceIndex(const Board *board, char pieceId);
bool canMove(const Board *board, const Piece *piece, char direction,
int steps);
void movePiece(Board *board, char pieceId, char direction, int
steps);



bool isSolved(const Board* board);
void freeBoard(Board *board); //free board and its pieces
```

```
Board* copyBoard(const Board* src);


#endif
```

- hashset.h

```
#ifndef HASHSET_H
#define HASHSET_H

#include "board.h"

#define HASHSET_SIZE 1000

// Struktur untuk elemen dalam HashSet
typedef struct HashSetEntry {
    Board* state;
    struct HashSetEntry* next;
} HashSetEntry;

// Struktur HashSet
typedef struct HashSet {
     HashSetEntry* table[HASHSET_SIZE];   // Array untuk menyimpan
entry
} HashSet;

HashSet* createHashSet();
void insertToHashSet(HashSet* set, Board* state);
bool containsInHashSet(HashSet* set, Board* state);
void freeHashSet(HashSet* set);

#endif
```

- pathfinder.h

```
// pathfinder.h
#ifndef PATHFINDER_H
#define PATHFINDER_H

#include "board.h"
#include "piece.h"
#include "utils.h"
#include "priorityqueue.h"
#include "hashset.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>


typedef struct Node {
    Board* state;
    struct Node* parent;
    Move move;
    int g; // Cost from start to this node
    int h; // Heuristic cost to goal
    int f; // Total cost (g + h)
} Node;

Node* createNode(Board* state, int g, int h,  Node* parent,Move*
move);

bool isStateExplored(HashSet* exploredStates, Board* state);;

void reconstructPath(Board* startNode, Node* goalNode);

int heuristic1(const Board *board);// Manhattan Distance to Exit

Board**     generateNextStates(const     Board     *board,     int
*nextStatesCount, Move* moves);
```

```
Node**   generateChildNodes(Node*   parent,   int*   count,   int
heuristicChoice);

int compareGBFS(const Node* a, const Node* b);

int compareUCS(const Node* a, const Node* b);

int compareAStar(const Node* a, const Node* b);
Node* genericSearch(Board initialBoard, int heuristicChoice, int
(*compare)(const Node*, const Node*)) ;

Node*      greedyBestFirstSearch(Board      initialBoard,      int
heuristicChoice);

Node* uniformCostSearch(Board initialBoard);

Node* aStarSearch(Board initialBoard, int heuristicChoice);


#endif
```

- piece.h

```
// piece.h
#ifndef PIECE_H
#define PIECE_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

typedef struct {
    char id;
    int row;
```

```
    int col;
    int size;
    char orientation; // H: horizontal, V: vertical
    int isPrimary; // 1 if primary piece, 0 otherwise

} Piece;



#endif
```

- priorityqueue.h

```
//priorityqueue.h
#ifndef PRIORITYQUEUE_H
#define PRIORITYQUEUE_H

#include "board.h"

typedef struct Node {
    Board* state;
    struct Node* parent;
    Move move; // Move that led to this state
    int g; // Cost from start to this node
    int h; // Heuristic cost to goal
    int f; // Total cost (g + h)
} Node;

typedef struct {
    Node** elements;
    int size;
    int capacity;
    int (*compare)(const Node*, const Node*);
} PriorityQueue;

PriorityQueue*  createPriorityQueue(int  (*compare)(const  Node*,
```

```
const Node*));

void enqueue(PriorityQueue* pq, Node* node);

Node* dequeue(PriorityQueue* pq);

void freePriorityQueue(PriorityQueue* pq);

int isEmpty(PriorityQueue* pq);
#endif
```

- utils.h

```
#ifndef UTILS_H
#define UTILS_H

#include "board.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>

void trim(char *str);
Board* loadBoardFromFile(const char *filename);
void printSolution(const Move *solution,int length);

bool    loadAndInitializeBoard(const    char    *filename,    Board
*initialBoard);
#endif
```

- main.c

```c
//main.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "board.h"
#include "utils.h"
#include "piece.h"
#include "priorityqueue.h"
#include "pathfinder.h"
#include "hashset.h"

#include <stdbool.h>
#include "time.h"


int main() {
    char filename[256];
    printf("Enter the input filename: ");
    scanf("%s", &filename);

    Board initialBoard;

    initBoard(&initialBoard);


    if (!loadAndInitializeBoard(filename, &initialBoard)) {
        printf("Failed to load and initialize board\n");
        return 1;
    }

    printf("Papan awal:\n");
    printBoard(&initialBoard);

    int algorithmChoice,heuristicChoice;

    printf("Choose an algorithm:\n");
    printf("1. Greedy Best-First Search\n");
```

```
    printf("2. Uniform Cost Search\n");
    printf("3. A* Search\n");
    scanf("%d", &algorithmChoice);

   heuristicChoice = 1;



    clock_t t1, t2;
    t1 = clock();
    printf("==========================\n");
    //====algorithm
    Node* goalNode = NULL;



    switch (algorithmChoice) {
        case 1:
                    printf("Running Greedy Best-First Search with
heuristic %d...\n", heuristicChoice);
                    goalNode = greedyBestFirstSearch(initialBoard,
heuristicChoice);
            break;
        case 2:
            printf("Running Uniform Cost Search...\n");
            goalNode = uniformCostSearch(initialBoard);
            break;
        case 3:
                printf("Running A* Search with heuristic %d...\n",
heuristicChoice);
            goalNode = aStarSearch(initialBoard, heuristicChoice);
            break;
        default:
            printf("Invalid algorithm choice.\n");

            return 1;
    }

    //=====

    t2 = clock();
```

```
    double waktu_ms = (double)(t2 - t1) * 1000.0 / CLOCKS_PER_SEC;


    if (goalNode != NULL) {
        printf("Solusi Ditemukan:\n\n");
        reconstructPath(&initialBoard, goalNode);

        // Count the number of moves in the solution`
        int moveCount = 0;
        Node* current = goalNode;
        while (current->parent != NULL) {
            moveCount++;
            current = current->parent;
        }

        printf("Jumlah gerakan yang diperiksa %d\n", totalCheck);
              printf("Banyaknya  gerakan  yang  dibutuhkan:  %d\n",
moveCount);

        // Free memory for the solution path
        current = goalNode;
        while (current != NULL) {
            Node* temp = current;
            current = current->parent;
            if (temp->state) {
                freeBoard(temp->state);
            }
            free(temp);
        }
        printf("\n");
        printf("Waktu Eksekusi: %.2f ms\n", waktu_ms);


    } else {
        printf("Solusi tidak ditemukan.\n");
        printf("Jumlah gerakan yang diperiksa %d\n", totalCheck);

        printf("Banyaknya gerakan: 0\n");
```

```
    }

    freeBoard(&initialBoard);
    return 0;

}
```

# Analisis & Pengujian

## 4.1. Hasil Percobaan

### 4.1.1 Percobaan 1 (tester1.txt)

Ketika exit berada di atas

Dengan menggunakan algoritma GBFS

```
==========================
Running Greedy Best-First Search with heuristic 1...
Greedy Best-First Search with heuristic 1
Solusi Ditemukan:

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

Gerakan:
Gerakan 1: Piece 'B' geser ke Kanan sebanyak 1 steps
    K
A A . B . F
. . P C D F
G . P C D F
G H . I I I
G H J . . .
L L J M M .

Jumlah gerakan yang diperiksa 156
Banyaknya gerakan yang dibutuhkan: 1

Waktu Eksekusi: 1.00 ms
```

Dengan menggunakan algortima UCS:

```
==========================
Running Uniform Cost Search...
Solusi Ditemukan:

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

Gerakan:
Gerakan 1: Piece 'B' geser ke Kanan sebanyak 1 steps
    K
A A . B . F
. . P C D F
G . P C D F
G H . I I I
G H J . . .
L L J M M .

Jumlah gerakan yang diperiksa 156
Banyaknya gerakan yang dibutuhkan: 1

Waktu Eksekusi: 1.00 ms
```

Dengan menggunakan algoritma A*:

```
==========================
Running A* Search with heuristic 1...
Solusi Ditemukan:

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

Gerakan:
Gerakan 1: Piece 'B' geser ke Kanan sebanyak 1 steps
    K
A A . B . F
. . P C D F
G . P C D F
G H . I I I
G H J . . .
L L J M M .

Jumlah gerakan yang diperiksa 156
Banyaknya gerakan yang dibutuhkan: 1

Waktu Eksekusi: 0.00 ms
```

4.1.2 Percobaan 2 (tester2.txt)
Ketika exit berada di samping kiri
Dengan GBFS

```
==========================
Running Greedy Best-First Search with heuristic 1...
Greedy Best-First Search with heuristic 1
Solusi Ditemukan:

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


Gerakan:
Gerakan 1: Piece 'C' geser ke Atas sebanyak 1 steps
  A A B C . F
  . . B C D F
K G P P . D F
  G H . I I I
  G H J . . .
  L L J M M .
```

```
Gerakan 2: Piece 'G' geser ke Atas sebanyak 1 steps
  A A B C . F
  G . B C D F
K G P P . D F
  G H . I I I
  . H J . . .
  L L J M M .


Gerakan 3: Piece 'J' geser ke Atas sebanyak 1 steps
  A A B C . F
  G . B C D F
K G P P . D F
  G H J I I I
  . H J . . .
  L L . M M .


Gerakan 4: Piece 'M' geser ke Kiri sebanyak 1 steps
  A A B C . F
  G . B C D F
K G P P . D F
  G H J I I I
  . H J . . .
  L L M M . .
```

```
Gerakan 5: Piece 'C' geser ke Bawah sebanyak 1 steps
  A A B . . F
  G . B C D F
K G P P C D F
  G H J I I I
  . H J . . .
  L L M M . .


Gerakan 6: Piece 'M' geser ke Kanan sebanyak 2 steps
  A A B . . F
  G . B C D F
K G P P C D F
  G H J I I I
  . H J . . .
  L L . . M M


Gerakan 7: Piece 'L' geser ke Kanan sebanyak 1 steps
  A A B . . F
  G . B C D F
K G P P C D F
  G H J I I I
  . H J . . .
  . L L . M M
```

```
Gerakan 8: Piece 'M' geser ke Kiri sebanyak 1 steps
  A A B . . F
  G . B C D F
K G P P C D F
  G H J I I I
  . H J . . .
  . L L M M .


Gerakan 9: Piece 'G' geser ke Bawah sebanyak 2 steps
  A A B . . F
  . . B C D F
K . P P C D F
  G H J I I I
  G H J . . .
  G L L M M .


Jumlah gerakan yang diperiksa 13173
Banyaknya gerakan yang dibutuhkan: 9

Waktu Eksekusi: 3.00 ms
```

Dengan UCS:

```
==========================
Running Uniform Cost Search...
Solusi Ditemukan:

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


Gerakan:
Gerakan 1: Piece 'J' geser ke Atas sebanyak 1 steps
  A A B . . F
  . . B C D F
K G P P C D F
  G H J I I I
  G H J . . .
  L L . M M .


Gerakan 2: Piece 'L' geser ke Kanan sebanyak 1 steps
  A A B . . F
  . . B C D F
K G P P C D F
  G H J I I I
  G H J . . .
  . L L M M .
```

```
Gerakan 3: Piece 'G' geser ke Bawah sebanyak 1 steps
  A A B . . F
  . . B C D F
K . P P C D F
  G H J I I I
  G H J . . .
  G L L M M .


Jumlah gerakan yang diperiksa 6499
Banyaknya gerakan yang dibutuhkan: 3

Waktu Eksekusi: 3.00 ms
```

Dengan A*:

```
Gerakan:
Gerakan 1: Piece 'J' geser ke Atas sebanyak 1 steps
  A A B . . F
  . . B C D F
K G P P C D F
  G H J I I I
  G H J . . .
  L L . M M .


Gerakan 2: Piece 'L' geser ke Kanan sebanyak 1 steps
  A A B . . F
  . . B C D F
K G P P C D F
  G H J I I I
  G H J . . .
  . L L M M .


Gerakan 3: Piece 'G' geser ke Bawah sebanyak 1 steps
  A A B . . F
  . . B C D F
K . P P C D F
  G H J I I I
  G H J . . .
  G L L M M .


Jumlah gerakan yang diperiksa 6045
Banyaknya gerakan yang dibutuhkan: 3
```

4.1.3 Percobaan 3 (tester3.txt)
Percobaan sesuai spesifikasi
Dengan GBFS heuristic 1:

```
Running Greedy Best-First Search with heuristic 1...
Greedy Best-First Search with heuristic 1
Solusi Ditemukan:

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 .


Gerakan:
Gerakan 1: Piece 'C' geser ke Atas sebanyak 1 steps
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: Piece 'P' geser ke Kanan sebanyak 1 steps
A A B C . F
. . B C D F
G . P P D F K
G H . I I I
G H J . . .
L L J M M .
```

```
Gerakan 3: Piece 'D' geser ke Atas sebanyak 1 steps
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: Piece 'P' geser ke Kanan sebanyak 1 steps
A A B C D F
. . B C D F
G . . P P F K
G H . I I I
G H J . . .
L L J M M .


Gerakan 5: Piece 'I' geser ke Kiri sebanyak 1 steps
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 6: Piece 'F' geser ke Bawah sebanyak 3 steps
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


Jumlah gerakan yang diperiksa 2764
Banyaknya gerakan yang dibutuhkan: 6

Waktu Eksekusi: 1.00 ms
```

Dengan UCS:

```
Running Uniform Cost Search...
Solusi Ditemukan:

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 .


Gerakan:
Gerakan 1: Piece 'D' geser ke Atas sebanyak 1 steps
A A B . D F
. . B C D F
G P P C . F K
G H . I I I
G H J . . .
L L J M M .


Gerakan 2: Piece 'C' geser ke Atas sebanyak 1 steps
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: Piece 'I' geser ke Kiri sebanyak 1 steps
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: Piece 'F' geser ke Bawah sebanyak 3 steps
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


Jumlah gerakan yang diperiksa 16530
Banyaknya gerakan yang dibutuhkan: 4

Waktu Eksekusi: 5.00 ms
```

Dengan A*:

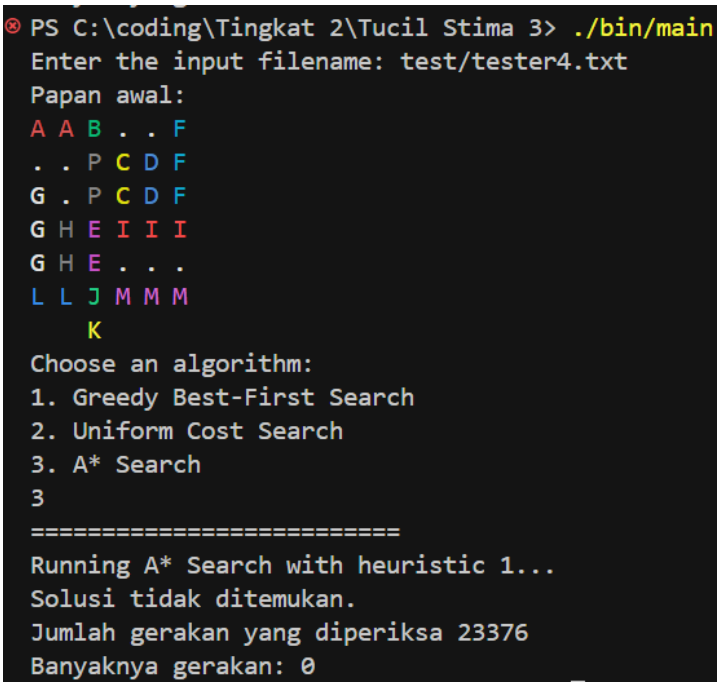
```
Running A* Search with heuristic 1...
Solusi Ditemukan:

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 .


Gerakan:
Gerakan 1: Piece 'D' geser ke Atas sebanyak 1 steps
A A B . D F
. . B C D F
G P P C . F K
G H . I I I
G H J . . .
L L J M M .


Gerakan 2: Piece 'C' geser ke Atas sebanyak 1 steps
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: Piece 'P' geser ke Kanan sebanyak 2 steps
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: Piece 'I' geser ke Kiri sebanyak 1 steps
A A B C D F
. . B C D F
G . . P P F K
G H I I I .
G H J . . .
L L J M M .


Gerakan 5: Piece 'F' geser ke Bawah sebanyak 3 steps
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


Jumlah gerakan yang diperiksa 9279
Banyaknya gerakan yang dibutuhkan: 5

Waktu Eksekusi: 4.00 ms
```

4.1.4 Percobaan 4 (tester4.txt)
Ketika tidak mungkin ada solusi

```
PS C:\coding\Tingkat 2\Tucil Stima 3> ./bin/main
Enter the input filename: test/tester4.txt
Papan awal:
A A B . . F
. . P C D F
G . P C D F
G H E I I I
G H E . . .
L L J M M M
    K
Choose an algorithm:
1. Greedy Best-First Search
2. Uniform Cost Search
3. A* Search
1
==========================
Running Greedy Best-First Search with heuristic 1...
Greedy Best-First Search with heuristic 1
Solusi tidak ditemukan.
Jumlah gerakan yang diperiksa 23376
Banyaknya gerakan: 0
```

```
PS C:\coding\Tingkat 2\Tucil Stima 3> ./bin/main
Enter the input filename: test/tester4.txt
Papan awal:
A A B . . F
. . P C D F
G . P C D F
G H E I I I
G H E . . .
L L J M M M
    K
Choose an algorithm:
1. Greedy Best-First Search
2. Uniform Cost Search
3. A* Search
2
==========================
Running Uniform Cost Search...
Solusi tidak ditemukan.
Jumlah gerakan yang diperiksa 23376
Banyaknya gerakan: 0
```


PS C:\coding\Tingkat 2\Tucil Stima 3> ./bin/main
Enter the input filename: test/tester4.txt
Papan awal:
A A B . . F
. . P C D F
G . P C D F
G H E I I I
G H E . . .
L L J M M M
K
Choose an algorithm:
1. Greedy Best-First Search
2. Uniform Cost Search
3. A* Search
3
==========
Running A* Search with heuristic 1...
Solusi tidak ditemukan.
Jumlah gerakan yang diperiksa 23376
Banyaknya gerakan: 0

4.1.5. Percobaan 5 (tester5.txt)

Teka tekin tersulit menurut Michael Fogleman

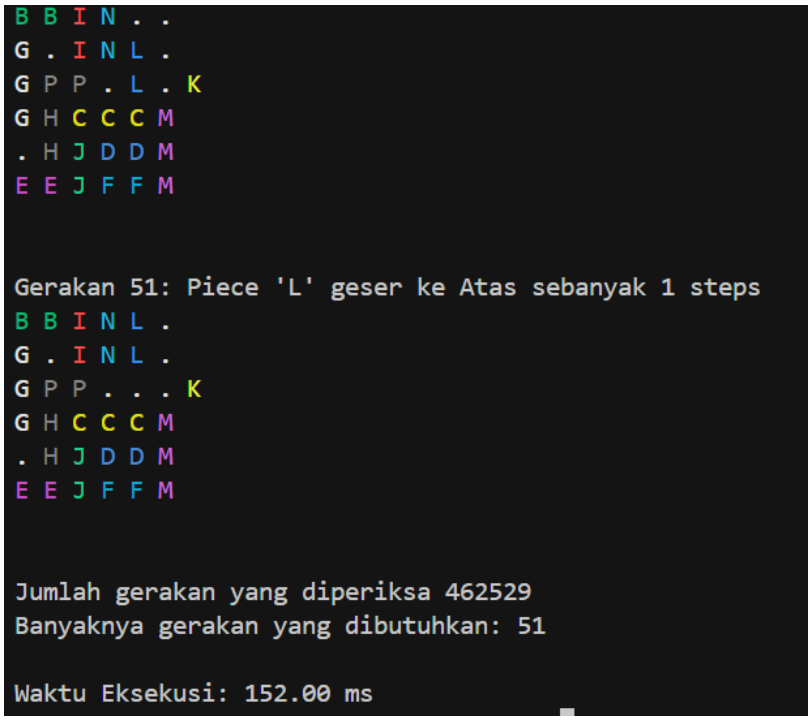
```
PS C:\coding\Tingkat 2\Tucil Stima 3> ./bin/main
Enter the input filename: test/tester6.txt
Papan awal:
G B B . L .
G H I . L M
G H I P P M K
C C C N . M
. . J N D D
E E J F F .

Choose an algorithm:
1. Greedy Best-First Search
2. Uniform Cost Search
3. A* Search
```

Dengan GBFS:

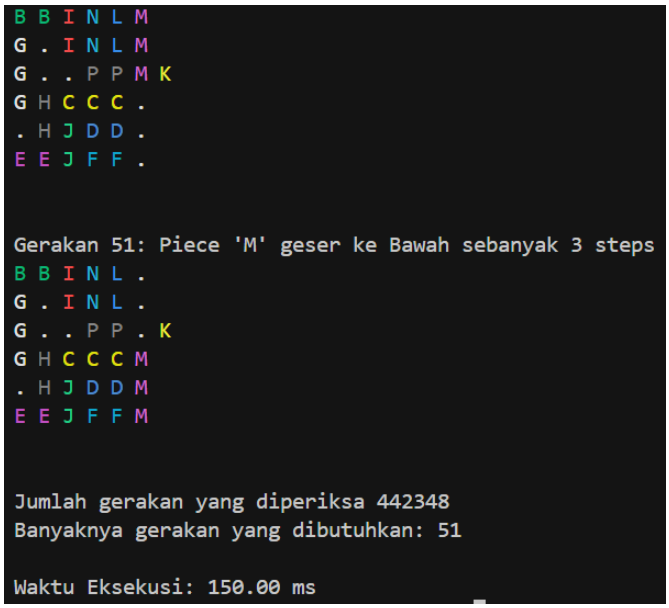
Jumlah gerakan akan sangat banyak dan melebihi kapasitas alokasi dinamis sehingga stop di step 195

```
Gerakan 194: Piece 'F' geser ke Kiri sebanyak 1 steps
. B B N L .
. . I N L M
G . I P P M K
G H C C C M
G H J D D .
E E J F F .


Gerakan 195: Piece 'M' geser ke Bawah sebanyak 2 steps
. B B N L .
. . I N L .
G . I P P . K
G H C C C M
G H J D D M
E E J F F M
```

Dengan UCS:
Hasil didapatkan dengan 51 step




Dengan A*:



## 4.2. Hasil dan Analisis Percobaan

**Hasil percobaan 1 :**

| | GBFS | UCS | A* |
|---|---|---|---|
| Jumlah check gerakan | 156 | 156 | 156 |
| Banyak hasil gerakan | 1 | 1 | 1 |
| Waktu (ms) | 1 | 1 | 0 |

Pada percobaan pertama, ketiga algoritma berjalan dengan alur dan cara yang sama dapat dilihat dari hasil percobaan tersebut. Percobaan ini cukup sederhana karena membutuhkan hanya 1 pergerakan untuk dapat mengeluarkan primary piece.

Hasil percobaan 2 :

| | GBFS | UCS | A* |
|---|---|---|---|
| Jumlah check gerakan | 13173 | 6499 | 6045 |
| Banyak hasil gerakan | 9 | 3 | 3 |
| Waktu (ms) | 3 | 3 | 3 |

Pada percobaan kedua, GBFS melakukan pemeriksaan gerakan yang jauh lebih banyak dan menghasilkan solusi yang tidak optimal. UCS dapat menghasilkan solusi yang optimal namun pengecekan masih lebih besar dibanding A*. Algoritma A* dapat memiliki jumlah pengecekan lebih kecil dari UCS karena memperhitungkan estimasi jarak ke exit dan juga berhasil memiliki solusi optimal.

Hasil percobaan 3 :

| | GBFS | UCS | A* |
|---|---|---|---|
| Jumlah check gerakan | 2764 | 16530 | 9279 |
| Banyak hasil gerakan | 6 | 4 | 5 |
| Waktu (ms) | 1 | 5 | 4 |

Pada percobaan 3, algoritma A* melakukan pengecekan yang lebih sedikit daripada UCS karena heuristic yang membantu cara bergerak. Hasil solusi yang terdapat UCS memliki yang paling efisien namun memiliki waktu yang paling besar dibanding GBFS dan UCS.

Hasil percobaan 4 :

| | GBFS | UCS | A* |
|---|---|---|---|
| Jumlah check gerakan | 23376 | 23376 | 23376 |
| Banyak hasil gerakan | 0 | 0 | 0 |
| Waktu (ms) | - | - | - |

Pada percobaan 4 memiliki kasus khusus supaya dapat menunjukkan jika terdapat kasus yang tidak dapat menghasikan solusi. Ketiga algoritma tersebut melakukan penelusuran yang sama meskipun path yang berbeda.

Hasil percobaan 5 :

| | GBFS | UCS | A* |
|---|---|---|---|
| Jumlah check gerakan | - | 462529 | 442348 |
| Banyak hasil gerakan | >195 | 51 | 51 |
| Waktu (ms) | - | 152 | 150 |

Pada percobaan 5, percobaan ini dapat dikategorikan kasus sulit. Algoritma GBFS memeriksa sangat banyak gerakan dan dapat menghasilkan solusi. Dengan algoritma GBFS menghasilkan solusi yang kurang efektif dibanding UCS dan A* hingga melebihi alokasi memory sehingga hasil tidak terprint sampai akhir. Dengan algortima UCS dan A* memiliki solusi yang paling efektif. Algoritma A* berhasil mengecek dengan jumlah yang lebih sedikit dibanding UCS dan waktu yang dibutuhkan lebih kecil dibanding UCS.

Secara umum, dari percobaan-percobaan ini dapat disimpulkan, algoritma GBFS memiliki waktu eksekusi yang paling cepat namun tidak dapat dipastikan menghasilkan solusi yang paling singkat. Pada kasus tertentu, GBFS memiliki jumlah pengecekan yang lebih banyak dibanding algoritma yang lain. Namun pada kasus tertentu, heuristic berhasil mempersingkat gerakan. GBFS menghasilkan solusi yang kurang optimal karena GBFS tidak memperhitungkan cost yang telah dikeluarkan.

Algoritma UCS selalu berhasil menghasilkan solusi yang paling optimal. Namun algoritma UCS perlu melakukan pengecekan yang lebih banyak dibanding algoritma lainnya karena algoritma ini tidak melakukan estimasi dalam gerakan dengan heuristic yang dapat mempersingkat pengecekan. Serta dari segi waktu eksekusi, UCS memerlukan waktu yang paling lama karena node yang diperiksa lebih banyak.

Algoritma A* menggunakan prinsip GBFS dan UCS secara bersamaan sehingga algoritma ini berhasil memiliki solusi yang optimal. Algoritma A* dapat menghitung cost yang telah dipakai dan dapat

mempersingkat pengecekan jalur dengan heuristic. Dari sisi waktu, A* dapat diproses lebih cepat dibandingan UCS, namun tidak secepat GBFS. Algoritma ini lebih cepat dibanding UCS karena A* mengecek jalur dengan lebih efektif, namun A* lebih lama dibanding GBFS karena GBFS tidak menghitung cost jalur yang ditempuh. Dapat disimpulkan algoritma A* merupakan algoritma lebih efektif dan efisien dibanding kedua algoritma lainnya.

# Kesimpulan

Berdasarkan analisis dan implementasi yang telah dilakukan, dapat disimpulkan bahwa algoritma pencarian pathfinding seperti Greedy Best-First Search, Uniform Cost Search, dan A* Search merupakan metode yang efektif untuk menyelesaikan permainan puzzle Rush Hour. Algoritma-algoritma ini bekerja dengan mengeksplorasi ruang state papan permainan, menghasilkan kemungkinan pergerakan kendaraan, dan mencari urutan pergerakan optimal yang mengarah pada keluarnya mobil utama dari papan.

Algoritma GBFS lebih cepat, namun sering kali tidak menghasilkan solusi optimal dan bisa mengeksplorasi jalur yang tidak relevan. Sedangkan algorita UCS dapat menjamin solusi optimal, tetapi membutuhkan waktu dan memori yang lebih banyak karena eksplorasi lebih menyeluruh. Sedangkan algoritma A* lebih efisien dan lebih terarah dalam menemukan solusi optimal, menggabungkan manfaat dari UCS dan GBFS, sehingga lebih cocok untuk masalah yang lebih kompleks.

Secara keseluruhan, algoritma pathfinding yang paling efektif dan efisien untuk digunakan pada game Rush Hour adalah algoritma A*.

# Lampiran

## Link Github

https://github.com/StefanMattew/Tucil3_13523020.git

## Tabel Check

| 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 | ✅ | |