Laporan Tugas Kecil 3
IF2211 Strategi Algoritma

Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding

Disusun oleh:

Rendi Adinata (10123083)
Muhammad Naufal Rayhannida (10123006)

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

BAB I
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.

Komponen penting dari permainan Rush Hour terdiri dari:

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

Pada laporan ini akan dijelaskan implementasi algoritma UCS, Best First Search dan A* untuk menyelesaikan Rush Hour Puzzle.

BAB 2
Algoritma UCS, Greedy Best First Search, dan A*

Definisi f(n) dan g(n)

- g(n) adalah biaya terakumulasi dari simpul awal hingga simpul n, dalam Rush Hour dihitung sebagai jumlah langkah yang telah diambil
- f(n) adalah fungsi evaluasi total untuk simpul n, didefinisikan sebagai f(n)=g(n)+h(n), dengan h(n) adalah heuristik estimasi biaya tersisa ke tujuan. Dalam proyek ini, fungsi heuristik yang digunakan adalah jarak (Manhattan) antara bagian depan *primary piece* dan pintu keluar.

Apakah heuristik A* admissible?

Dikarenakan untuk setiap langkah *primary piece* hanya dapat dipindahkan sebanyak satu sel, heuristik ini tidak pernah melebih-lebihkan jumlah langkah minimal yang sebenarnya diperlukan. Dengan demikian, heuristik tersebut *admissible*, tidak melebihi biaya sesungguhnya.

Apakah UCS sama dengan BFS di Rush Hour?

Ya, urutan pembangkitan node dengan UCS dan BFS sama, karena UCS dengan biaya seragam 1 memiliki urutan ekspansi mirip BFS. Akan tetapi, struktur data berbeda (priority queue vs queue).

Apakah A* lebih efisien secara teori dibanding UCS?

Secara teoritis A* dengan heuristik admissible memandu pencarian menuju area menjanjikan dan menghindari eksplorasi simpul-simpul jauh dari solusi, sehingga meminimalkan jumlah ekspansi simpul dibandingkan UCS yang merupakan *uninformed search.*

Apakah Greedy Best-First menjamin solusi optimal?

Tidak. Karena Greedy Best-First Search hanya mempertimbangkan h(n) dan mengabaikan g(n). Algoritma ini dapat mengikuti jalur yang terlihat dekat dengan tujuan namun memiliki banyak *detour*. Oleh karena itu, algoritma ini tidak menjamin menemukan jalur dengan jumlah langkah paling sedikit

UCS (Uniformed Cost Search)

Dalam permainan Rush Hour, metode pencarian Uniform Cost Search (UCS) dimulai dengan memasukkan papan awal sebagai simpul akar dengan biaya awal g=0 . Semua simpul yang akan diekspansi disimpan dalam antrian prioritas yang diurutkan berdasarkan nilai g terkecil. Setiap iterasi, UCS mengambil simpul dengan biaya terkecil, memeriksa apakah posisi mobil utama (P) sudah mencapai pintu keluar, dan jika belum, menghasilkan semua gerakan valid berikutnya (*neighbors*). Setiap gerakan ditambahkan sebagai simpul baru dengan biaya g_baru= g_induk +1 , dan dimasukkan ke dalam antrian bila belum pernah dikunjungi. Karena UCS selalu mengekspansi jalur terendah biaya terlebih dahulu dan menjamin menemukan solusi optimal meski explorasi cenderung melebar seperti BFS.

Inisialisasi

1. Buat priority queue `pq` dengan komparator berdasarkan g (biaya terakumulasi).

2. Push status awal (`startBoard`) dengan `g=0`.

3. Tandai papan awal sebagai visited.

```
auto cmpUCS = [](const State &a, const State &b){
   return a.g > b.g;
};
priority_queue<State, vector<State>,
decltype(cmpUCS)> pq(cmpUCS);
pq.push({startBoard, {}, 0, 0});
visited.insert(boardToString(startBoard));
```

Loop Ekspansi

- Pop status dengan g terkecil.

- Periksa apakah itu merupakan kondisi kemenangan (mobil ‘P’ sudah di depan pintu).

- Jika belum, generate semua tetangga (`getNeighbors`)—yaitu semua pergeseran satu langkah—dan untuk setiap tetangga yang belum pernah dilihat, tambahkan ke `pq` dengan `g+1`.

```
while (!pq.empty()) {
    State cur = pq.top(); pq.pop();
    // cek kemenangan
    if (isGoal(cur)) { finalMoves = cur.moves; break; }
    // perluas tetangga
    for (auto &nxt : getNeighbors(cur)) {
        string key = boardToString(nxt.board);
        if (!visited.count(key)) {
            visited.insert(key);
            pq.push(nxt);
        }
    }
}
```

Output

Setelah menemukan solusi, urutan `cur.moves` adalah jalur optimal (minimal langkah) karena UCS memberikan solusi optimal pada graf berbiaya seragam.

Greedy Best First Search

Greedy Best-First Search hanya menggunakan fungsi heuristik h untuk memandu pencarian. Fungsi heuristik yang digunakan adalah jarak Manhattan antara bagian depan mobil dengan pintu keluar. Mula-mula, simpul akar diberi nilai h=`heuristic(startBoard)`, kemudian disimpan dalam antrian prioritas yang diurutkan berdasarkan nilai h terkecil. Pada setiap langkah, simpul dengan estimasi tersisa terkecil diambil, diperiksa sebagai goal, dan jika belum mencapai tujuan, semua tetangganya dihasilkan dan dinilai ulang heuristiknya sebelum dimasukkan. Karena fokus utamanya adalah mereduksi , bukan biaya sebenarnya , algoritma ini cepat menuruni jalur yang tampak menjanjikan, tetapi tidak menjamin optimalitas dan bisa terjebak pada jalur buntu.

Inisialisasi

1. Buat priority queue `pq` dengan komparator berdasarkan h(n) saja.
2. Push status awal dengan `g=0` dan `h = heuristic(startBoard)`.

3. Tandai papan awal sebagai visited.

```
auto cmpGreedy = [](const State &a, const State &b){
    return a.h > b.h;
};
priority_queue<State, vector<State>,
decltype(cmpGreedy)> pq(cmpGreedy);
pq.push({startBoard, {}, 0, heuristic(startBoard)});
visited.insert(boardToString(startBoard));
```

Loop Ekspansi

```
while (!pq.empty()) {
    State cur = pq.top(); pq.pop();
    if (isGoal(cur)) { finalMoves = cur.moves; break; }
    for (auto &nxt : getNeighbors(cur)) {
        string key = boardToString(nxt.board);
        if (!visited.count(key)) {
            visited.insert(key);
            nxt.h = heuristic(nxt.board);
            pq.push(nxt);
        }
    }
}
```

- Pop status dengan h terkecil.

- Periksa kondisi kemenangan.

- Generate tetangga, hitung `h(nxt)` untuk masing-masing, dan push hanya yang belum visited.

Perhatikan bahwa, karena tidak mempertimbangkan `g`, Greedy bisa masuk jalur buntu atau solusi non-optimal.

A*

A* mengombinasikan kelebihan UCS dan Greedy dengan menggunakan fungsi evaluasi f=g+h . Simpul akar dimulai dengan g=0 dan h sama seperti heuristik start, lalu seluruh simpul diprioritaskan berdasarkan nilai f terkecil. Ketika sebuah simpul diekspansi, tetangganya dihasilkan, g diupdate menjadi g+1, h dihitung ulang, dan f baru dimasukkan ke dalam antrian bila belum dikunjungi. Heuristik yang digunakan sama dengan Best First Search. Perhatikan bahwa fungsi heuristik ini admissible karena tidak akan overestimate sehingga berdasarkan teorema optimalitas A* terjamin .

Inisialisasi

```
auto cmpAstar = [](const State &a, const State &b){
   return (a.g + a.h) > (b.g + b.h);
};
priority_queue<State, vector<State>,
decltype(cmpAstar)> pq(cmpAstar);
pq.push({startBoard, {}, 0, heuristic(startBoard)});
visited.insert(boardToString(startBoard));
```

1. Buat priority queue `pq` dengan komparator berdasarkan f(n) = g + h.

2. Push status awal dengan `g=0`, `h=heuristic(startBoard)`.

3. Tandai papan awal sebagai visited.

Loop Ekspansi

```
while (!pq.empty()) {
    State cur = pq.top(); pq.pop();
    if (isGoal(cur)) { finalMoves = cur.moves; break; }
    for (auto &nxt : getNeighbors(cur)) {
        string key = boardToString(nxt.board);
        if (!visited.count(key)) {
            visited.insert(key);
            nxt.h = heuristic(nxt.board);
            // nxt.g sudah ditambahkan dalam getNeighbors
            pq.push(nxt);
        }
    }
}
```

- Pop status dengan nilai f(n) terkecil.
- Periksa kemenangan.
- Generate tetangga, untuk setiap `nxt`:
    - Hitung `nxt.h = heuristic(nxt.board)`
    - Biaya terakumulasi `nxt.g = cur.g + 1`
    - Jika belum visited, push ke `pq`.

Dengan heuristik admissible, A* hanya mengeksplorasi simpul dengan `f(n)` kurang dari solusi optimal, sehingga jauh lebih efisien dibanding UCS.

BAB 3

SOURCE CODE PROGRAM

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <queue>
#include <unordered_set>
#include <set>
#include <chrono>
#include <cmath>
#include <utility>
#include <algorithm>

using namespace std;
using Board = vector<string>;

struct State {
    Board board;
    vector<string> moves;
    size_t g; // Cost from start
    size_t h; // Heuristic estimate
};

int A, B, N;
pair<int, int> exit_pos;

string boardToString(const Board &b) {
    string s;
    for (const auto &row : b) s += row;
    return s;
}

void findPrimary(const Board &b, int &r, int &c, int &len, bool
&horizontal) {
    for (int i = 0; i < A; ++i) {
        for (int j = 0; j < B; ++j) {
            if (b[i][j] == 'P') {
```

```
                horizontal = (j + 1 < B && b[i][j + 1] == 'P');
                len = 1;
                if (horizontal) {
                    while (j + len < B && b[i][j + len] == 'P')
++len;
                } else {
                    while (i + len < A && b[i + len][j] == 'P')
++len;
                }
                r = i;
                c = j;
                return;
            }
        }
    }
}

int heuristic(const Board &b) {
    int r, c, len;
    bool horizontal;
    findPrimary(b, r, c, len, horizontal);
    int front = horizontal ? (c + len - 1) : (r + len - 1);
    int exit_dist;
    if (horizontal) {
        exit_dist = exit_pos.second - front - 1;
    } else {
        exit_dist = exit_pos.first - front - 1;
    }
    return max(0, exit_dist);  // Pastikan tidak negatif
}

vector<State> getNeighbors(const State &cur) {
    vector<State> res;
    const Board &b = cur.board;
    set<char> seen;

    for (int i = 0; i < A; ++i) {
        for (int j = 0; j < B; ++j) {
            char ch = b[i][j];
            if (ch == '.' || ch == 'K' || seen.count(ch))
continue;
            seen.insert(ch);
```

```
            vector<pair<int, int>> cells;
            for (int x = 0; x < A; ++x)
                for (int y = 0; y < B; ++y)
                    if (b[x][y] == ch) cells.emplace_back(x, y);

            bool horizontal = (cells[0].first == cells[1].first);
            int minr = A, maxr = 0, minc = B, maxc = 0;
            for (auto &p : cells) {
                minr = min(minr, p.first);
                maxr = max(maxr, p.first);
                minc = min(minc, p.second);
                maxc = max(maxc, p.second);
            }

            if (horizontal) {
                // Geser kiri
                int steps_left = 0;
                for (int y = minc - 1; y >= 0; --y) {
                    if (b[minr][y] != '.') break;
                    steps_left++;
                }
                if (steps_left > 0) {
                    Board nb = b;
                    for (int k = 0; k < (maxc - minc + 1); ++k)
                        nb[minr][minc + k] = '.';
                    for (int k = 0; k < (maxc - minc + 1); ++k)
                        nb[minr][minc - steps_left + k] = ch;
                    State ns{nb, cur.moves, cur.g + 1, 0};
                    ns.moves.push_back(string(1, ch) + "-L");
                    res.push_back(ns);
                }

                // Geser kanan
                int steps_right = 0;
                for (int y = maxc + 1; y < B; ++y) {
                    if (b[minr][y] != '.') break;
                    steps_right++;
                }
                if (steps_right > 0) {
                    Board nb = b;
                    for (int k = 0; k < (maxc - minc + 1); ++k)
```

```
                    nb[minr][minc + k] = '.';
                for (int k = 0; k < (maxc - minc + 1); ++k)
                    nb[minr][minc + steps_right + k] = ch;
                State ns{nb, cur.moves, cur.g + 1, 0};
                ns.moves.push_back(string(1, ch) + "-R");
                res.push_back(ns);
            }
        } else {
            // Geser atas
            int steps_up = 0;
            for (int x = minr - 1; x >= 0; --x) {
                if (b[x][minc] != '.') break;
                steps_up++;
            }
            if (steps_up > 0) {
                Board nb = b;
                for (int k = 0; k < (maxr - minr + 1); ++k)
                    nb[minr + k][minc] = '.';
                for (int k = 0; k < (maxr - minr + 1); ++k)
                    nb[minr - steps_up + k][minc] = ch;
                State ns{nb, cur.moves, cur.g + 1, 0};
                ns.moves.push_back(string(1, ch) + "-U");
                res.push_back(ns);
            }

            // Geser bawah
            int steps_down = 0;
            for (int x = maxr + 1; x < A; ++x) {
                if (b[x][minc] != '.') break;
                steps_down++;
            }
            if (steps_down > 0) {
                Board nb = b;
                for (int k = 0; k < (maxr - minr + 1); ++k)
                    nb[minr + k][minc] = '.';
                for (int k = 0; k < (maxr - minr + 1); ++k)
                    nb[minr + steps_down + k][minc] = ch;
                State ns{nb, cur.moves, cur.g + 1, 0};
                ns.moves.push_back(string(1, ch) + "-D");
                res.push_back(ns);
            }
        }
```

```
        }
    }
    return res;
}

string dirWord(char d) {
    switch (d) {
        case 'L': return "kiri";
        case 'R': return "kanan";
        case 'U': return "atas";
        case 'D': return "bawah";
        default: return "";
    }
}

int main() {
    string path;
    cout << "Masukkan file path input : ";
    getline(cin, path); // Baca seluruh line termasuk spasi

    ifstream fin(path);
    if (!fin) {
        cerr << "Cannot open " << path << "\n";
        perror("Error details");
        return 1;
    }

    fin >> A >> B >> N;
    Board startBoard(A);
    for (int i = 0; i < A; ++i) fin >> startBoard[i];
    fin.close();

    // Cari posisi exit (K)
    exit_pos = {-1, -1};
    for (int i = 0; i < A; ++i) {
        for (int j = 0; j < B; ++j) {
            if (startBoard[i][j] == 'K') {
                exit_pos = {i, j};
                break;
            }
        }
        if (exit_pos.first != -1) break;
```

```
    }

    if (exit_pos.first == -1) {
        cerr << "Exit (K) tidak ditemukan di papan!\n";
        return 1;
    }

    int alg;
    cout << "Pilih algoritma (1-UCS, 2-Best First Search, 3-A*):
";
    cin >> alg;

    auto startTime = chrono::high_resolution_clock::now();
    vector<string> finalMoves;
    unordered_set<string> visited;

    if (alg == 1) { // UCS
        auto cmp = [](const State &a, const State &b) { return a.g
> b.g; };
        priority_queue<State, vector<State>, decltype(cmp)>
pq(cmp);
        pq.push({startBoard, {}, 0, 0});
        visited.insert(boardToString(startBoard));

        while (!pq.empty()) {
            State cur = pq.top(); pq.pop();
            int r, c, len; bool horizontal;
            findPrimary(cur.board, r, c, len, horizontal);
            // Perbaikan kondisi kemenangan
            int front = horizontal ? (c + len - 1) : (r + len -
1);
            if ((horizontal && r == exit_pos.first && front ==
exit_pos.second - 1) ||
                (!horizontal && c == exit_pos.second && front ==
exit_pos.first - 1)) {
                finalMoves = cur.moves;
                break;
            }
            for (auto &nxt : getNeighbors(cur)) {
                string key = boardToString(nxt.board);
                if (!visited.count(key)) {
                    visited.insert(key);
```

```
                    pq.push(nxt);
                }
            }
        }
    } else { // Best-First atau A*
        auto cmp = [alg](const State &a, const State &b) {
            if (alg == 2) return a.h > b.h;
            return (a.g + a.h) > (b.g + b.h);
        };
        priority_queue<State, vector<State>, decltype(cmp)>
pq(cmp);
        State initial{startBoard, {}, 0,
(size_t)heuristic(startBoard)};
        pq.push(initial);
        visited.insert(boardToString(startBoard));

        while (!pq.empty()) {
            State cur = pq.top(); pq.pop();
            int r, c, len; bool horizontal;
            findPrimary(cur.board, r, c, len, horizontal);
            int front = horizontal ? (c + len - 1) : (r + len -
1);
            if ((horizontal && r == exit_pos.first && front ==
exit_pos.second - 1) ||
                (!horizontal && c == exit_pos.second && front ==
exit_pos.first - 1)) {
                finalMoves = cur.moves;
                break;
            }
            for (auto &nxt : getNeighbors(cur)) {
                string key = boardToString(nxt.board);
                if (!visited.count(key)) {
                    visited.insert(key);
                    nxt.h = heuristic(nxt.board);
                    pq.push(nxt);
                }
            }
        }
    }

    // Output hasil
    cout << "Papan Awal\n";
```

```
    for (auto &row : startBoard) cout << row << "\n";
    cout << "\n";

    if (finalMoves.empty()) {
        cout << "Tidak ada solusi ditemukan.\n";
    } else {
        Board current = startBoard;

        string base_filename =
path.substr(path.find_last_of("/\\") + 1);
        string::size_type const
p(base_filename.find_last_of('.'));
        string file_without_extension = base_filename.substr(0,
p);
        ofstream move_out(file_without_extension + "-moves.txt");
        move_out << A << ' ' << B << endl;
        for (auto s: startBoard) {
            move_out << s << endl;
        }

        for (size_t i = 0; i < finalMoves.size(); ++i) {
            string move = finalMoves[i];
            char car = move[0];
            char dir = move[2];
            cout << "Gerakan " << i + 1 << ": " << car << "-" <<
dirWord(dir) << "\n";
            bool found = false;
            vector<State> neighbors = getNeighbors({current, {},
0, 0});
            for (auto &n : neighbors) {
                if (!n.moves.empty() && n.moves.back() == move) {
                    current = n.board;
                    found = true;
                    break;
                }
            }
            if (!found) {
                cerr << "Error: Gerakan tidak valid: " << move <<
"\n";
                return 1;
            }
            for (auto &row : current) cout << row << "\n";
```

```
            cout << "\n";

            move_out << move << endl;
        }
    }

    auto endTime = chrono::high_resolution_clock::now();
    cout << "Waktu (ms): " <<
chrono::duration_cast<chrono::milliseconds>(endTime -
startTime).count() << "\n";
    return 0;
}
```

BAB 4
HASIL PENGUJIAN

Test case 1:

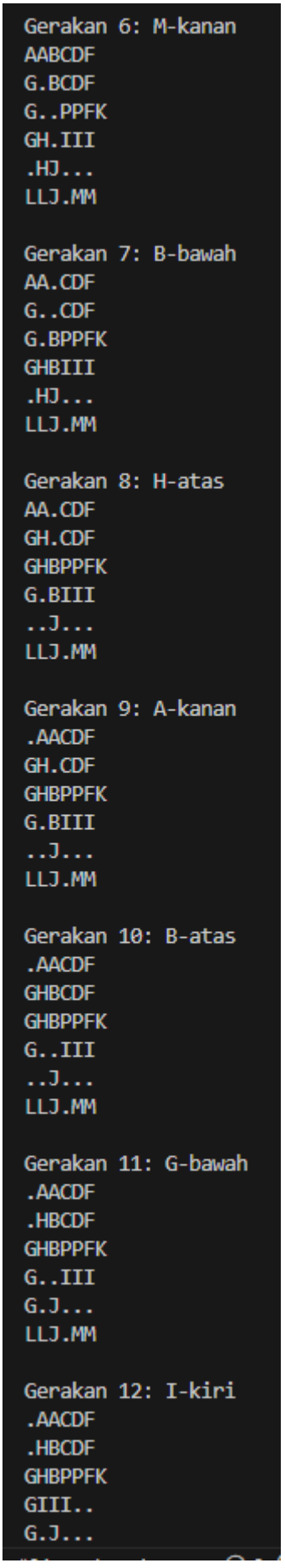
UCS

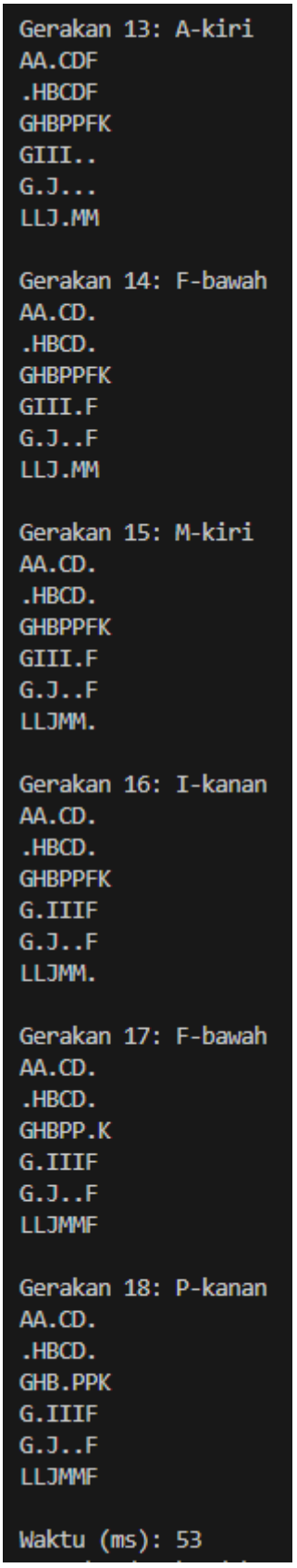
Masukan	Keluaran
6 6 12 AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.	Papan Awal AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM. Gerakan 1: C-atas AABC.F ..BCDF GPP.DFK GH.III GHJ... LLJMM. Gerakan 2: I-kiri AABC.F ..BCDF GPP.DFK GHIII. GHJ... LLJMM. Gerakan 3: F-bawah AABC.. ..BCD. GPP.D.K GHIIIF GHJ..F LLJMMF Gerakan 4: D-atas AABCD. ..BCD. GPP...K GHIIIF GHJ..F LLJMMF Gerakan 5: P-kanan AABCD. ..BCD. G...PPK GHIIIF GHJ..F LLJMMF Waktu (ms): 22

Masukan	Keluaran
	Waktu (ms): 20

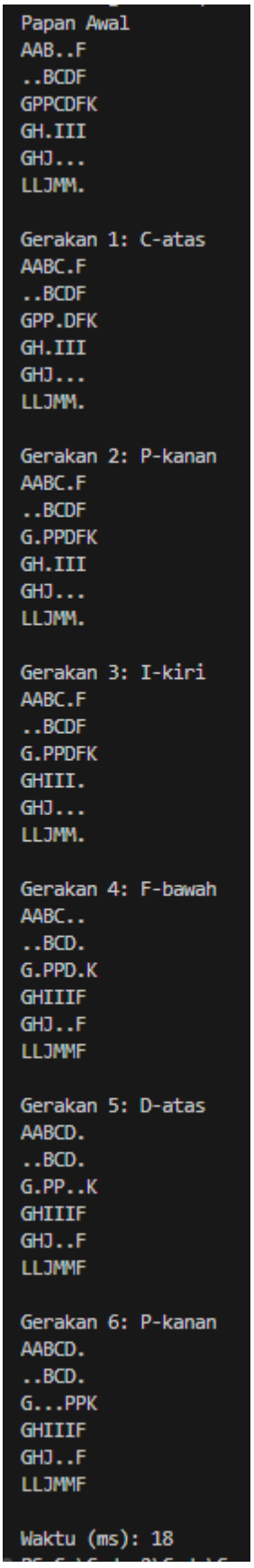
Best First Search

Masukan	Keluaran
	Papan Awal AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM. Gerakan 1: C-atas AABC.F ..BCDF GPP.DFK GH.III GHJ... LLJMM. Gerakan 2: P-kanan AABC.F ..BCDF G.PPDFK GH.III GHJ... LLJMM. Gerakan 3: D-atas AABCDF ..BCDF G.PP.FK GH.III GHJ... LLJMM. Gerakan 4: P-kanan AABCDF ..BCDF G..PPFK GH.III GHJ... LLJMM. Gerakan 5: G-atas AABCDF G.BCDF G..PPFK GH.III .HJ... LLJMM.

	 Gerakan 6: M-kanan AABCDF G.BCDF G..PPFK GH.III .HJ... LLJ.MM Gerakan 7: B-bawah AA.CDF G..CDF G.BPPFK GHBIII .HJ... LLJ.MM Gerakan 8: H-atas AA.CDF GH.CDF GHBPPFK G.BIII ..J... LLJ.MM Gerakan 9: A-kanan .AACDF GH.CDF GHBPPFK G.BIII ..J... LLJ.MM Gerakan 10: B-atas .AACDF GHBCDF GHBPPFK G..III ..J... LLJ.MM Gerakan 11: G-bawah .AACDF .HBCDF GHBPPFK G..III G.J... LLJ.MM Gerakan 12: I-kiri .AACDF .HBCDF GHBPPFK GIII.. G.J...

Masukan	Keluaran
	

A*

Masukan	Keluaran
	

Test case 2:

UCS

Masukan	Keluaran

```
7 8
10
A.P....B
A.PCC..B
AFFH..GG
...H....
I.JJJ...
I.......
.OO.....
  K
```

```
Papan Awal
A.P....B
A.PCC..B
AFFH..GG
...H....
I.JJJ...
I.......
.OO.....
  K

Gerakan 1: J-kanan
A.P....B
A.PCC..B
AFFH..GG
...H....
I....JJJ
I.......
.OO.....
  K

Gerakan 2: H-bawah
A.P....B
A.PCC..B
AFF...GG
........
I....JJJ
I..H....
.OOH....
  K

Gerakan 3: F-kanan
A.P....B
A.PCC..B
A...FFGG
........
I....JJJ
I..H....
.OOH....
  K

Gerakan 4: O-kiri
A.P....B
A.PCC..B
A...FFGG
........
I....JJJ
I..H....
OO.H....
  K

Gerakan 5: P-bawah
A......B
A..CC..B
A...FFGG
........
I....JJJ
I.PH....
OOPH....
  K

Waktu (ms): 57
```

Best First Search

Masukan	Keluaran

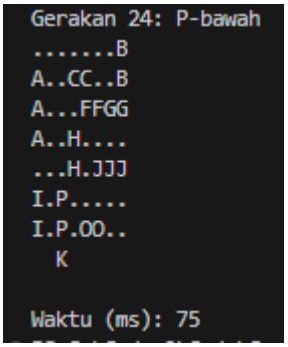
```
7 8
10
A.P....B
A.PCC..B
AFFH..GG
...H....
I.JJJ...
I.......
.OO.....
  K
```

```
Papan Awal
A.P....B
A.PCC..B
AFFH..GG
...H....
I.JJJ...
I.......
.OO.....
  K

Gerakan 1: A-bawah
..P....B
A.PCC..B
AFFH..GG
A..H....
I.JJJ...
I.......
.OO.....
  K

Gerakan 2: O-kanan
..P....B
A.PCC..B
AFFH..GG
A..H....
I.JJJ...
I.......
......OO
  K

Gerakan 3: G-kiri
..P....B
A.PCC..B
AFFHGG..
A..H....
I.JJJ...
I.......
......OO
  K

Gerakan 4: O-kiri
..P....B
A.PCC..B
AFFHGG..
A..H....
I.JJJ...
I.......
OO......
  K

Gerakan 5: J-kanan
..P....B
A.PCC..B
AFFHGG..
A..H....
I....JJJ
I.......
OO......
  K
```

```
Gerakan 6: H-bawah
..P....B
A.PCC..B
AFF.GG..
A.......
I....JJJ
I..H....
OO.H....
  K

Gerakan 7: F-kanan
..P....B
A.PCC..B
A.FFGG..
A.......
I....JJJ
I..H....
OO.H....
  K

Gerakan 8: O-kanan
..P....B
A.PCC..B
A.FFGG..
A.......
I....JJJ
I..H....
.OOH....
  K

Gerakan 9: H-atas
..P....B
A.PCC..B
A.FFGG..
A..H....
I..H.JJJ
I.......
.OO.....
  K

Gerakan 10: O-kanan
..P....B
A.PCC..B
A.FFGG..
A..H....
I..H.JJJ
I.......
......OO
  K

Gerakan 11: H-bawah
..P....B
A.PCC..B
A.FFGG..
A.......
I....JJJ
I..H....
...H..OO
  K
```

```
Gerakan 12: O-kiri
..P....B
A.PCC..B
A.FFGG..
A.......
I....JJJ
I..H....
...HOO..
  K

Gerakan 13: H-atas
..P....B
A.PCC..B
A.FFGG..
A..H....
I..H.JJJ
I.......
....OO..
  K

Gerakan 14: J-kiri
..P....B
A.PCC..B
A.FFGG..
A..H....
I..HJJJ.
I.......
....OO..
  K

Gerakan 15: I-bawah
..P....B
A.PCC..B
A.FFGG..
A..H....
...HJJJ.
I.......
I...OO..
  K

Gerakan 16: O-kiri
..P....B
A.PCC..B
A.FFGG..
A..H....
...HJJJ.
I.......
IOO.....
  K

Gerakan 17: G-kanan
..P....B
A.PCC..B
A.FF..GG
A..H....
...HJJJ.
I.......
IOO.....
  K
```

	Gerakan 18: O-kanan
	..P....B
	A.PCC..B
	A.FF..GG
	A..H....
	...HJJJ.
	I.......
	I.....OO
	  K
	
	Gerakan 19: J-kanan
	..P....B
	A.PCC..B
	A.FF..GG
	A..H....
	...H.JJJ
	I.......
	I.....OO
	  K
	
	Gerakan 20: H-bawah
	..P....B
	A.PCC..B
	A.FF..GG
	A.......
	.....JJJ
	I..H....
	I..H..OO
	  K
	
	Gerakan 21: O-kiri
	..P....B
	A.PCC..B
	A.FF..GG
	A.......
	.....JJJ
	I..H....
	I..HOO..
	  K
	
	Gerakan 22: H-atas
	..P....B
	A.PCC..B
	A.FF..GG
	A..H....
	...H.JJJ
	I.......
	I...OO..
	  K
	
	Gerakan 23: F-kanan
	..P....B
	A.PCC..B
	A...FFGG
	A..H....
	...H.JJJ
	I.......
	I...OO..
	  K

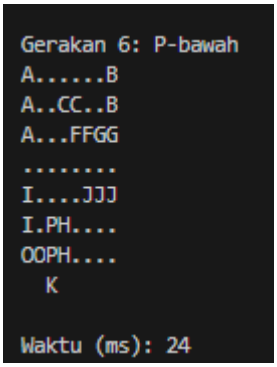
Masukan	Keluaran
	

A*

Masukan	Keluaran

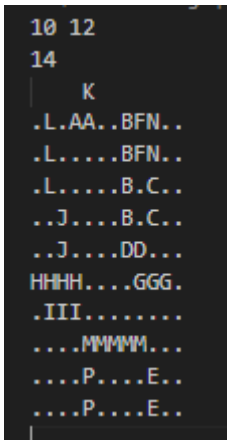
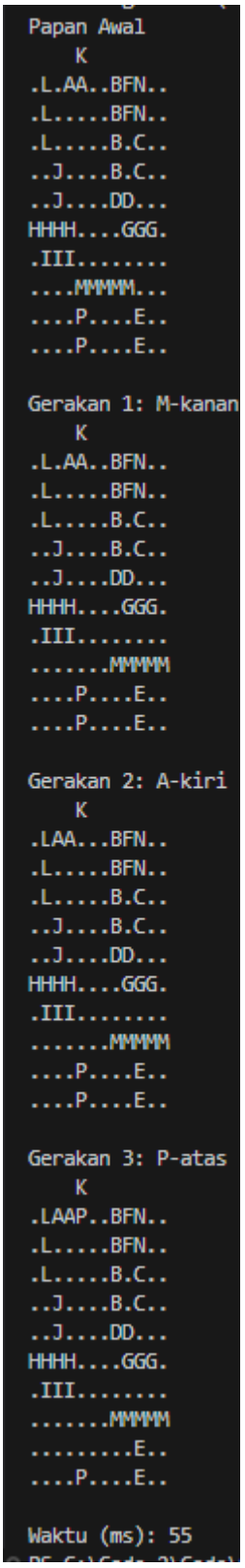
```
7 8
10
A.P....B
A.PCC..B
AFFH..GG
...H....
I.JJJ...
I.......
.OO.....
  K
```

```
Papan Awal
A.P....B
A.PCC..B
AFFH..GG
...H....
I.JJJ...
I.......
.OO.....
  K

Gerakan 1: J-kanan
A.P....B
A.PCC..B
AFFH..GG
...H....
I....JJJ
I.......
.OO.....
  K

Gerakan 2: H-bawah
A.P....B
A.PCC..B
AFF...GG
........
I....JJJ
I..H....
.OOH....
  K

Gerakan 3: F-kanan
A.P....B
A.PCC..B
A...FFGG
........
I....JJJ
I..H....
.OOH....
  K

Gerakan 4: P-bawah
A......B
A..CC..B
A...FFGG
........
I.P..JJJ
I.PH....
.OOH....
  K

Gerakan 5: O-kiri
A......B
A..CC..B
A...FFGG
........
I.P..JJJ
I.PH....
OO.H....
  K
```

Masukan	Keluaran
	

Test case 3:

UCS

Masukan	Keluaran
	

Best First Search

Masukan	Keluaran
10 12 14 K .L.AA..BFN.. .L.....BFN.. .L.....B.C.. ..J....B.C.. ..J....DD... HHHH....GGG. .III..... ....MMMM... ....P....E.. ....P....E..	Gerakan 37: N-atas K .....NAA .L......FN.. .L......F... .LJ....B.C.. ..J..DDB.C.. HHHH...BGGG. .III...B.... ....MMMM... ....P....E.. ....P....E.. Gerakan 38: C-atas K .....NAA .L......FN.. .L......FC.. .LJ....B.C.. ..J..DDB.... HHHH...BGGG. .III...B.... ....MMMM... ....P....E.. ....P....E.. Gerakan 39: M-kanan K .....NAA .L......FN.. .L......FC.. .LJ....B.C.. ..J..DDB.... HHHH...BGGG. .III...B.... .....MMMM ....P....E.. ....P....E.. Gerakan 40: P-atas K ....P....NAA .L......FN.. .L......FC.. .LJ....B.C.. ..J..DDB.... HHHH...BGGG. .III...B.... .....MMMM .....E.. ....P....E.. Waktu (ms): 201

A*

Masukan	Keluaran

```
10 12
14
   K
.L.AA..BFN..
.L.....BFN..
.L.....B.C..
..J....B.C..
..J....DD...
HHHH....GGG.
.III........
....MMMMM...
....P....E..
....P....E..
```

```
Papan Awal
   K
.L.AA..BFN..
.L.....BFN..
.L.....B.C..
..J....B.C..
..J....DD...
HHHH....GGG.
.III........
....MMMMM...
....P....E..
....P....E..

Gerakan 1: M-kanan
   K
.L.AA..BFN..
.L.....BFN..
.L.....B.C..
..J....B.C..
..J....DD...
HHHH....GGG.
.III........
.......MMMM
....P....E..
....P....E..

Gerakan 2: P-atas
   K
.L.AA..BFN..
.L..P..BFN..
.L.....B.C..
..J....B.C..
..J....DD...
HHHH....GGG.
.III........
.......MMMM
.........E..
....P....E..

Gerakan 3: A-kiri
   K
.LAA...BFN..
.L..P..BFN..
.L.....B.C..
..J....B.C..
..J....DD...
HHHH....GGG.
.III........
.......MMMM
.........E..
....P....E..

Gerakan 4: P-atas
   K
.LAAP..BFN..
.L.....BFN..
.L.....B.C..
..J....B.C..
..J....DD...
HHHH....GGG.
.III........
.......MMMM
.........E..
```

Masukan	Keluaran
	Waktu (ms): 22

Test case 4:

UCS

Masukan	Keluaran
7 12 10 .H....C..... .H....CBBBBB III...C...A. PP.......GA.K .EDDDD...GA. .E......... FFFF........	Pilih algoritma (1: Papan Awal .H....C..... .H....CBBBBB III...C...A. PP.......GA.K .EDDDD...GA. .E......... FFFF........ Gerakan 1: A-bawah .H....C..... .H....CBBBBB III...C..... PP.......G..K .EDDDD...GA. .E........A. FFFF......A. Gerakan 2: G-bawah .H....C..... .H....CBBBBB III...C..... PP.........K .EDDDD....A. .E.......GA. FFFF.....GA. Gerakan 3: P-kanan .H....C..... .H....CBBBBB III...C..... ....PPK .EDDDD....A. .E.......GA. FFFF.....GA. Waktu (ms): 16

Best First Search

Masukan	Keluaran

```
7 12
10
.H....C.....
.H....CBBBBB
III...C...A.
PP.......GA.K
.EDDDD...GA.
.E..........
FFFF........
```

```
Papan Awal
.H....C.....
.H....CBBBBB
III...C...A.
PP.......GA.K
.EDDDD...GA.
.E..........
FFFF........

Gerakan 1: P-kanan
.H....C.....
.H....CBBBBB
III...C...A.
.......PPGA.K
.EDDDD...GA.
.E..........
FFFF........

Gerakan 2: C-bawah
.H..........
.H.....BBBBB
III.......A.
.......PPGA.K
.EDDDDC..GA.
.E....C.....
FFFF..C.....

Gerakan 3: F-kanan
.H..........
.H.....BBBBB
III.......A.
.......PPGA.K
.EDDDDC..GA.
.E....C.....
..FFFFC.....

Gerakan 4: C-atas
.H....C.....
.H....CBBBBB
III...C...A.
.......PPGA.K
.EDDDD...GA.
.E..........
..FFFF......

Gerakan 5: D-kanan
.H....C.....
.H....CBBBBB
III...C...A.
.......PPGA.K
.E...DDDDGA.
.E..........
..FFFF......

Gerakan 6: G-bawah
.H....C.....
.H....CBBBBB
III...C...A.
.......PP.A.K
.E...DDDD.A.
.E.......G..
..FFFF...G..
```

Masukan	Keluaran
	Gerakan 7: P-kanan .H....C..... .H....CBBBBB III...C...A. ....PPA.K .E...DDDD.A. .E.......G.. ..FFFF...G.. Gerakan 8: A-bawah .H....C..... .H....CBBBBB III...C..... ....PP..K .E...DDDD.A. .E.......GA. ..FFFF...GA. Gerakan 9: P-kanan .H....C..... .H....CBBBBB III...C..... ....PPK .E...DDDD.A. .E.......GA. ..FFFF...GA. Waktu (ms): 28

A*

Masukan	Keluaran
7 12 10 .H....C..... .H....CBBBBB III...C...A. PP.......GA.K .EDDDD...GA. .E......... FFFF........	Papan Awal .H....C..... .H....CBBBBB III...C...A. PP.......GA.K .EDDDD...GA. .E......... FFFF........ Gerakan 1: P-kanan .H....C..... .H....CBBBBB III...C...A. ....PPGA.K .EDDDD...GA. .E......... FFFF........ Gerakan 2: G-bawah .H....C..... .H....CBBBBB III...C...A. .......PP.A.K .EDDDD....A. .E.......G.. FFFF.....G.. Gerakan 3: P-kanan .H....C..... .H....CBBBBB III...C...A. ....PPA.K .EDDDD....A. .E.......G.. FFFF.....G.. Gerakan 4: A-bawah .H....C..... .H....CBBBBB III...C..... ....PP..K .EDDDD....A. .E.......GA. FFFF.....GA. Gerakan 5: P-kanan .H....C..... .H....CBBBBB III...C..... ....PPK .EDDDD....A. .E.......GA. FFFF.....GA. Waktu (ms): 39

Bab 5

Analisis dan Pembahasan

Berdasarkan hasil pengujian, algoritma A* terbukti lebih efisien dibandingkan Uniform Cost Search (UCS) dan Greedy Best-First Search. Hal ini disebabkan oleh sifat A* yang menggabungkan keunggulan keduanya, yaitu mempertimbangkan biaya aktual (g) dan estimasi jarak ke tujuan (h). A* mampu menekan eksplorasi simpul yang tidak menjanjikan, sehingga menghasilkan solusi dengan jumlah langkah yang lebih sedikit dan waktu yang lebih cepat.

Sebaliknya, Greedy Best-First Search tidak selalu menghasilkan solusi optimal. Pada pengujian dengan test case ke-3, algoritma ini menghasilkan jalur dengan 40 langkah yang tidak efisien. Hal ini terjadi karena Greedy hanya berfokus pada nilai heuristik (h) dan mengabaikan biaya sebenarnya (g), sehingga mudah terjebak dalam jalur yang tampak dekat ke tujuan tetapi justru berputar-putar (detour).

Dalam hal kompleksitas, ketiga algoritma memiliki batas atas eksponensial $O(b^d)$ pada kasus terburuk, di mana b adalah faktor percabangan (branching factor) dan d adalah kedalaman solusi. UCS menelusuri semua simpul berdasarkan biaya terkecil, sehingga kompleksitasnya bertumbuh cepat terhadap ukuran solusi. Greedy Best-First Search juga berpotensi menjelajah banyak simpul jika heuristiknya tidak informatif. A* lebih efisien jika heuristik yang digunakan bersifat admissible (tidak melebih-lebihkan), karena hanya mengekspansi simpul yang benar-benar menjanjikan. Namun, jika heuristik lemah atau nol, perilaku A* akan menyerupai UCS.

REFERENSI

- artificial intelligence - What is the difference between uniform-cost search and best-first search methods? - Stack Overflow
- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf
- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf
- https://www.geeksforgeeks.org/breadth-first-search-is-a-special-case-of-uniform-cost-search/

LAMPIRAN

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	✅	

pranala *repository*:
https://github.com/xinuzo/Tucil3_10123006_10123083.git