Laporan Tugas Kecil 1 IF2211 Strategi Algoritma
IQ Puzzle Pro Solver

Muhammad Fathur Rizky
13523105
13523105@std.stei.itb.ac.id

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

Daftar Isi

Permasalahan

Permainan **IQ Puzzler Pro** merupakan permainan papan yang mengharuskan pemain untuk menyusun blok-blok puzzle ke dalam papan hingga terisi penuh. Setiap blok memiliki bentuk unik dan harus digunakan dalam penyelesaian puzzle tanpa adanya tumpang tindih, kecuali dalam mode permainan 3D. Pemain dapat melakukan rotasi maupun pencerminan pada blok untuk menemukan posisi yang tepat. Tantangan utama dalam permainan ini adalah bagaimana cara menyusun semua blok dalam keterbatasan ruang papan yang tersedia.

Dalam menyelesaikan permainan, terdapat beberapa aspek penting yang harus diperhatikan sebagai berikut.

- Papan/*board* permainan sebagai area utama yang harus diisi dengan blok secara penuh.
- Blok puzzle yang memiliki berbagai bentuk unik dan harus digunakan seluruhnya dalam penyelesaian.
- Rotasi dan pencerminan yang dapat dilakukan untuk menambah ruang penelusuran.
- Strategi penyusunan yang diperlukan agar tidak ada ruang kosong atau kesalahan dalam penempatan.

Pendekatan Solusi

Pendekatan paling sederhana untuk menyelesaikan IQ Puzzler Pro adalah menggunakan metode bruteforce, yaitu mencoba semua kemungkinan penempatan blok hingga ditemukan solusi yang memenuhi syarat. Pendekatan ini dapat direpresentasikan sebagai pencarian ruang solusi dengan eksplorasi semua kemungkinan peletakan blok dalam papan permainan.

Algoritma ini akan mencoba menempatkan blok satu per satu. Jika tidak memungkinkan, akan kembali ke langkah sebelumnya dan mencoba konfigurasi lain. Pendekatan ini memastikan bahwa setiap solusi yang ditemukan adalah solusi valid, tanpa ada tumpang tindih blok yang tidak diperbolehkan.

Implementasi dalam Kode

Struktur Data

```
import java.util.*;

public class Pair<T, U> {
    public T fi;
    public U se;

    public Pair(T x, U y) {
```

```
        this.fi = x;
        this.se = y;
    }
}

public class Tuple5 {
    public int N;
    public int M;
    public int P;
    public int[][] board;
    public List<Pair<Character, List<List<List<Integer>>>>> piece;

    public Tuple5(int N, int M, int[][] board, int P, List<Pair<Character,
List<List<List<Integer>>>>> piece) {
        this.N = N;
        this.M = M;
        this.board = board;
        this.P = P;
        this.piece = piece;
    }
}
```

Struktur data dalam kode ini terdiri dari dua kelas utama, yaitu Pair dan Tuple5 yang berfungsi untuk menyimpan data permainan IQ Puzzler Pro. Pair adalah kelas generik sederhana yang digunakan untuk menyimpan dua elemen dengan tipe yang berbeda. Dengan adanya Pair, kita bisa mengelompokkan dua data yang saling berkaitan dalam satu objek, misalnya karakter blok puzzle dan daftar bentuknya.

Sementara itu, Tuple5 menyimpan informasi utama permainan dalam satu struktur data. Kelas ini memiliki lima atribut, yaitu N dan M yang merepresentasikan ukuran papan (jumlah baris dan kolom), serta P menunjukkan jumlah total blok puzzle. Selain itu, ada board, yaitu matriks dua dimensi yang menggambarkan kondisi papan—baik yang kosong (0), sudah terisi blok (1–26), atau bagian yang tidak bisa digunakan (-1). Terakhir, piece adalah daftar pasangan karakter blok dan semua bentuknya, termasuk kemungkinan rotasi dan pencerminan.

Mencari Posisi Kosong pada Papan

```
private int[] fz(int[][] b) {
    int rs = b.length;
    int cs = b[0].length;
    for (int i = 0; i < rs; i++) {
        for (int j = 0; j < cs; j++) {
            if (b[i][j] == 0)
                return new int[] { i, j };
        }
    }
    return null;
}
```

Metode fz digunakan untuk menemukan posisi pertama yang kosong (bernilai 0) pada papan permainan. Fungsi ini menerima parameter berupa matriks dua dimensi b yang merepresentasikan kondisi papan permainan. Pertama, variabel rs dan cs diinisialisasi untuk menyimpan jumlah baris dan kolom papan. Kemudian, dilakukan iterasi melalui setiap sel papan menggunakan dua loop. Jika ditemukan sel dengan nilai 0, maka metode ini segera mengembalikan koordinat sel tersebut dalam bentuk array integer {i, j}, dengan i adalah indeks baris dan j adalah indeks kolom. Jika seluruh papan sudah terisi dan tidak ada lagi sel kosong, metode ini mengembalikan null yang menandakan bahwa tidak ada ruang kosong tersisa.

Memeriksa Apakah Blok Bisa Ditempatkan

```
private boolean isSafe(int[][] b, int x, int y, List<List<Integer>> v) {
    int rs = v.size();
    int cs = v.get(0).size();
    int br = b.length;
    int bc = b[0].length;
    for (int i = 0; i < rs; i++) {
        for (int j = 0; j < cs; j++) {
            if (v.get(i).get(j) == 1) {
                int bx = x + i;
                int by = y + j;
                if (bx < 0 || bx >= br || by < 0 || by >= bc || b[bx][by] !=
0) {
                    return false;
                }
            }
        }
    }
    return true;
}
```

Metode isSafe berfungsi untuk memeriksa apakah sebuah blok dapat ditempatkan pada posisi tertentu di papan permainan tanpa melanggar batas atau menimpa sel yang sudah terisi. Fungsi ini menerima parameter b, yaitu matriks papan permainan, serta x dan y yang merepresentasikan posisi relatif awal di mana blok akan ditempatkan. Selain itu, parameter v adalah representasi dari bentuk blok yang akan diletakkan dalam bentuk matriks dua dimensi.

Proses pengecekan diawali dengan mengambil ukuran blok rs (jumlah baris) dan cs (jumlah kolom). Selanjutnya, variabel br dan bc menyimpan dimensi papan permainan. Fungsi ini kemudian melakukan iterasi pada setiap elemen blok v. Jika ditemukan elemen dengan nilai 1, maka koordinat absolut pada papan permainan dihitung dengan menambahkan indeks iterasi i dan j ke posisi awal x dan y, menghasilkan koordinat (bx, by).

Jika koordinat tersebut berada di luar batas papan (bx atau by melebihi ukuran papan atau kurang dari nol) atau sudah terisi (b[bx][by] != 0), maka metode ini mengembalikan false yang menandakan bahwa blok tidak dapat ditempatkan di lokasi tersebut. Jika seluruh elemen blok dapat ditempatkan tanpa pelanggaran, maka metode mengembalikan true, menandakan bahwa posisi tersebut aman untuk diletakkan.

Menempatkan Blok pada Papan

```
private void pl(int[][] b, int x, int y, List<List<Integer>> v, Character n)
{
    int rs = v.size();
    int cs = v.get(0).size();
    for (int i = 0; i < rs; i++) {
        for (int j = 0; j < cs; j++) {
            if (v.get(i).get(j) == 1) {
                b[x + i][y + j] = n - 'A' + 1;
            }
        }
    }
}
```

Metode pl berfungsi untuk menempatkan sebuah blok pada papan permainan dengan mengubah nilai sel yang sesuai berdasarkan bentuk blok yang diberikan. Parameter b adalah matriks papan permainan, sedangkan x dan y merupakan posisi awal tempat blok akan diletakkan. Parameter v adalah representasi bentuk variasi blok dalam matriks dua dimensi, dan n adalah karakter yang merepresentasikan blok tersebut.

Metode ini pertama-tama menentukan ukuran blok dengan mengambil jumlah baris (rs) dan jumlah kolom (cs) dari matriks v. Kemudian, dilakukan iterasi pada setiap elemen dalam blok. Jika suatu elemen dalam v bernilai 1, maka koordinatnya diterjemahkan ke posisi yang sesuai di papan dengan menambahkan indeks baris dan kolom iterasi ke x dan y. Sel tersebut kemudian diisi dengan nilai numerik yang diperoleh dari konversi karakter n menjadi angka, dengan perhitungan n - 'A' + 1. Konversi ini memastikan bahwa setiap blok memiliki representasi angka yang unik berdasarkan urutan alfabetnya.

Solusi Menggunakan Stack

```
private static class Pos {
    int[][] b;
    boolean[] vis;
    int rem;

    public Pos(int[][] b, boolean[] u, int rem) {
        this.b = b;
        this.vis = u;
        this.rem = rem;
    }
}
```

Pertama-tama, didefinisikan sebuah kelas Pos yang berfungsi untuk menyimpan state/posisi/*state* papan permainan pada suatu titik dalam proses pencarian solusi. Kelas ini memiliki tiga atribut utama yang merepresentasikan kondisi permainan.

Atribut pertama adalah b, yaitu sebuah matriks dua dimensi yang menggambarkan state papan permainan saat ini. Setiap elemen dalam matriks ini menunjukkan apakah suatu posisi di papan

masih kosong atau sudah terisi oleh sebuah blok. Selanjutnya, ada atribut vis, yaitu sebuah array boolean yang mencatat blok mana saja yang sudah digunakan dalam konfigurasi tersebut. Jika suatu blok sudah ditempatkan di papan, maka nilai pada indeks yang sesuai di array ini akan menjadi true.

Terakhir, atribut rem digunakan untuk menyimpan jumlah blok yang masih tersisa dan belum ditempatkan di papan. Ketika jumlah ini mencapai nol, artinya semua blok sudah diletakkan dan solusi telah ditemukan.

```
public void solve() {
    Stack<Pos> s = new Stack<>();
    int[][] ib = cpy(b);
    boolean[] iv = vis.clone();
    Pos is = new Pos(ib, iv, p);
    s.push(is);
    Pos ss = null;

    while (!s.isEmpty()) {
        Pos curr = s.pop();
        int[] n = fz(curr.b);
        if (n == null) {
            if (curr.rem == 0) {
                ss = curr;
                f = true;
                break;
            }
            continue;
        }
        int x = n[0];
        int y = n[1];
        for (var pi : ps) {
            Character ch = pi.fi;
            if (!curr.vis[ch - 'A']) {
                boolean[] nv = curr.vis.clone();
                nv[ch - 'A'] = true;
                for (List<List<Integer>> v : pi.se) {
                    int rs = v.size();
                    int cs = v.get(0).size();
                    int bx = x, by = y;
                    boolean fl = false;
                    for (int i = 0; i < rs; i++) {
                        for (int j = 0; j < cs; j++) {
                            it++;
                            if (v.get(i).get(j) == 1) {
                                bx = x - i;
                                by = y - j;
                                fl = true;
                                break;
                            }
                        }
                        if (fl)
                            break;
                    }
```

```
                    if (fl && isSafe(curr.b, bx, by, v)) {
                        int[][] nb = cpy(curr.b);
                        pl(nb, bx, by, v, ch);
                        Pos ns = new Pos(nb, nv, curr.rem - 1);
                        s.push(ns);
                    }
                }
            }
        }
    }
    if (ss != null) {
        b = ss.b;
    }
}
```

Fungsi solve() merupakan inti dari algoritma pencarian solusi untuk menyusun blok di papan permainan. Pendekatan yang digunakan adalah pencarian berbasis backtracking dengan bantuan stack sebagai struktur data utama untuk menyimpan dan mengelola setiap kemungkinan konfigurasi papan.

Pertama-tama, sebuah stack berisi objek Pos dibuat untuk menyimpan state papan yang perlu diperiksa. state awal permainan di-*copy* dari papan utama ke dalam variabel ib, sementara status penggunaan blok juga di-*copy* ke iv. Kemudian, state awal ini dimasukkan ke dalam stack untuk diproses. Variabel ss disiapkan untuk menyimpan solusi jika ditemukan.

Selama stack masih memiliki elemen, algoritma akan mengambil state papan teratas menggunakan pop(). Kemudian, algoritma mencari sel kosong pertama pada papan dengan memanggil fungsi fz(). Jika tidak ada sel kosong yang ditemukan, dan jumlah blok yang tersisa adalah nol, berarti solusi telah ditemukan. Pada titik ini, konfigurasi papan saat ini disimpan ke dalam ss, dan pencarian dihentikan. Jika masih ada sel kosong, untuk setiap iterasi dilakukan percobaan penempatan berbagai jenis blok yang tersedia di posisi tersebut.

Untuk setiap blok yang belum digunakan, algoritma mencoba berbagai kemungkinan orientasi dari blok tersebut. Sebelum menempatkan blok, diperiksa terlebih dahulu apakah blok tersebut bisa dipasang tanpa melanggar batas atau bertabrakan dengan blok lain menggunakan fungsi isSafe(). Jika sebuah blok bisa ditempatkan, maka papan permainan diperbarui dengan menambahkan blok tersebut, status penggunaan blok diperbarui, dan jumlah blok yang tersisa dikurangi. state baru ini kemudian dimasukkan kembali ke dalam stack sehingga bisa diperiksa nanti.

Analisis Kompleksitas

Analisis kompleksitas dari fungsi solve() bergantung pada jumlah kemungkinan konfigurasi papan yang harus diperiksa dalam pencarian solusi. Karena algoritma ini menggunakan penelusuran dengan stack sebagai struktur data utama, kompleksitasnya sangat dipengaruhi oleh jumlah blok yang harus ditempatkan dan ukuran papan permainan.

Misalkan P adalah jumlah blok yang tersedia, dan setiap blok memiliki beberapa kemungkinan rotasi atau bentuk. Jika kita anggap ada R adalah jumlah variasi untuk setiap blok, maka total semua variasi adalah RP. Selain itu, untuk setiap konfigurasi, algoritma melakukan pencarian sel kosong dengan fz() yang memiliki kompleksitas $O(NM)$ untuk papan berukuran N × M. Kemudian, pemeriksaan validitas penempatan blok menggunakan isSafe() juga memiliki kompleksitas $O(NM)$ dalam skenario terburuk.

Secara keseluruhan, kompleksitas waktu dari algoritma ini dapat diperkirakan sebagai $O(NM^{RP})$ jika memang sangat naif dalam pemecahan solusinya. Pada fungsi solve(), banyak dilakukan *pruning* sehingga kompleksitas riilnya jauh dari yang diperhitungkan. Hal itu dapat terlihat bahwa fungsi solve melakukan penelusuran jauh dari NM karena pengisian board dimulai dari pojok kiri atas dan dicari pada kasus valid saja.

Extras

Image Output

```
public void genImg(String file) {
    file += ".png";
    int cs = 50;
    int w = c * cs;
    int h = r * cs;
    // ARGB untuk mendukung transparansi.
    BufferedImage img = new BufferedImage(w, h, BufferedImage.TYPE_INT_ARGB);
    Graphics2D g = img.createGraphics();

    g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            if (b[i][j] != -1) {
                g.setColor(Color.GRAY);
                g.fillRect(j * cs, i * cs, cs, cs);
                g.setColor(Color.BLACK);
                g.drawRect(j * cs, i * cs, cs, cs);
            }
        }
    }

    // Gambarkan huruf pada cell (jika nilai board > 0).
    Font f = new Font("SansSerif", Font.BOLD, 24);
    g.setFont(f);
    FontMetrics m = g.getFontMetrics();

    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            if (b[i][j] > 0) {
```

```
                String l = String.valueOf((char) (b[i][j] + 'A' - 1));
                int tw = m.stringWidth(l);
                int th = m.getHeight();
                int x = j * cs + (cs - tw) / 2;
                int y = i * cs + ((cs - th) / 2) + m.getAscent();
                g.setColor(getColor(l.charAt(0)));
                g.drawString(l, x, y);
            }
        }
    }
    g.dispose(); // Tutup objek Graphics2D.

    try {
        ImageIO.write(img, "png", new File(file));
        System.out.println("Gambar disimpan di " + file);
    } catch (IOException e) {
        System.out.println("Error: pembuatan gambar gagal");
    }
}
```

Algoritma di atas bertujuan untuk menghasilkan sebuah gambar representasi dari papan permainan berdasarkan matriks yang diberikan. Pertama, nama file ditambahkan ekstensi ".png", kemudian ukuran gambar dihitung berdasarkan jumlah kolom dan baris, dengan setiap sel memiliki ukuran tetap sebesar 50 piksel. Gambar dibuat menggunakan objek BufferedImage dengan tipe ARGB agar mendukung transparansi. Selanjutnya, objek Graphics2D digunakan untuk menggambar elemen-elemen pada gambar, dengan mengaktifkan fitur anti-aliasing agar tampilan lebih halus.

Algoritma kemudian melakukan iterasi melalui setiap sel dalam matriks papan. Jika sebuah sel tidak memiliki nilai -1, maka sel tersebut dianggap sebagai bagian dari papan dan akan digambar sebagai persegi abu-abu dengan garis tepi hitam. Setelah menggambar grid papan, algoritma melanjutkan dengan menampilkan huruf pada sel yang memiliki nilai lebih dari nol. Huruf yang ditampilkan dihitung berdasarkan nilai numerik pada matriks, diubah ke karakter dengan menyesuaikan indeks huruf A sebagai dasar.

TXT Output

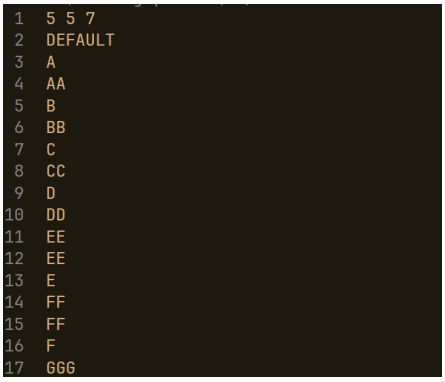
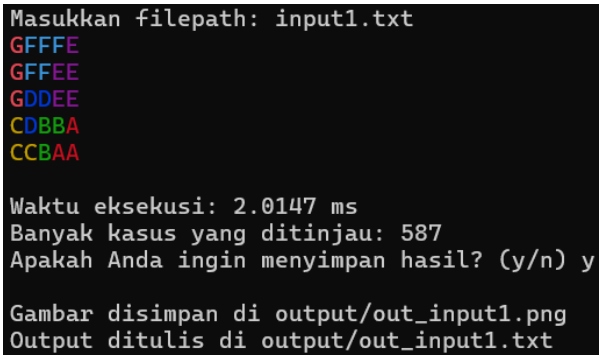
```
public void writeOutput(int iter, int[][] board, double exTime) {
    try {
        String fileName = "output/out_" + fn;
        File file = new File(fileName);
        try (Formatter formatter = new Formatter(fileName)) {

            for (int i = 0; i < board.length; i++) {
                for (int j = 0; j < board[i].length; j++) {
                    if (board[i][j] == -1) {
                        formatter.format(" ");
                    } else if (board[i][j] == 0) {
                        formatter.format("?");
```

```
                    } else {
                        formatter.format("%c", (char) ('A' + board[i][j] -
1));
                    }
                }
                formatter.format("\n");
            }
            formatter.format("\nWaktu Eksekusi: %.3f ms\n", exTime);
            formatter.format("Banyak kasus ditinjau: %d", iter);
        }
        System.out.println("Output ditulis di " + fileName);
    } catch (Exception e) {
        System.out.println("Error: gagal menulis output.");
        e.printStackTrace();
    }
}
```

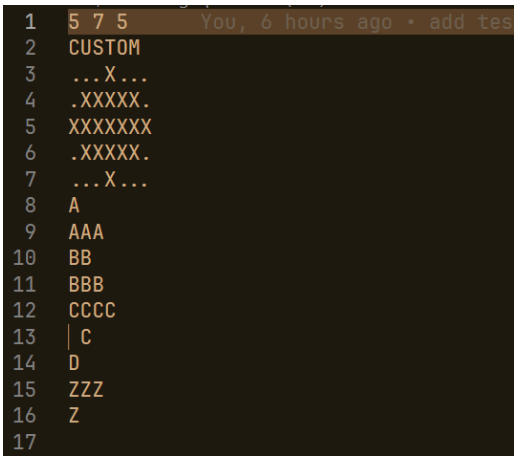
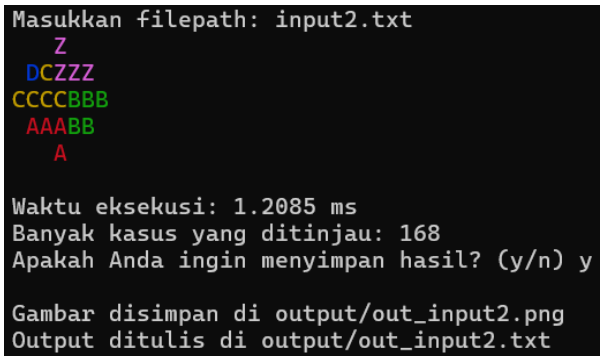
Algoritma ini menulis hasil solusi papan permainan ke dalam file txt dengan nama yang diawali "output/out_" diikuti nilai variabel fn. Program membuat file dan menggunakan Formatter untuk menulis isi papan. Setiap elemen matriks diperiksa, di mana nilai -1 ditulis sebagai spasi, 0 ditulis sebagai tanda tanya, dan angka lainnya dikonversi menjadi huruf berdasarkan nilai ASCII. Setelah seluruh papan ditulis, program menambahkan informasi waktu eksekusi dalam milidetik serta jumlah kasus yang telah ditinjau. Jika proses berhasil, pesan sukses ditampilkan, sedangkan jika terjadi kesalahan, program mencetak pesan error.

Pengujian Test Case

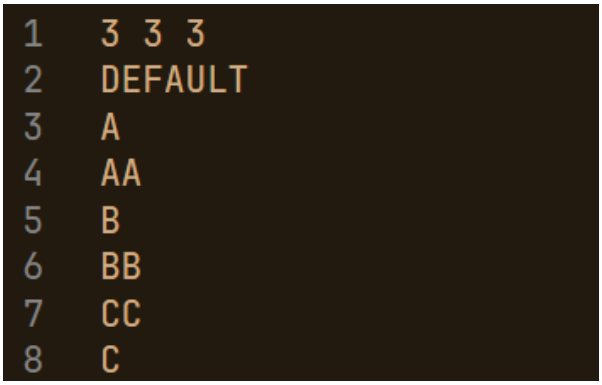
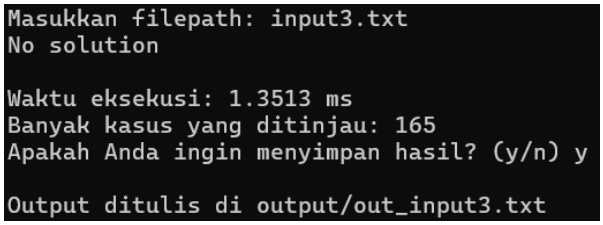
Test Case #1 (Default: Has Solution)

Input	Output
	

Test Case #2 (Custom: Has Solution)

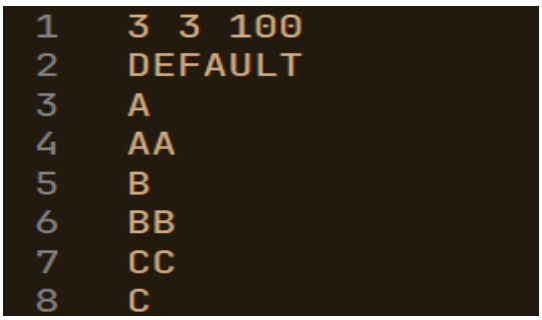
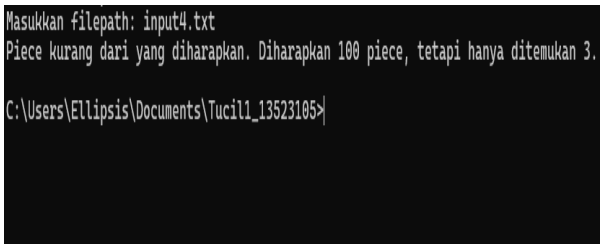
Input	Output
	

Test Case #3 (Default: No Solution)

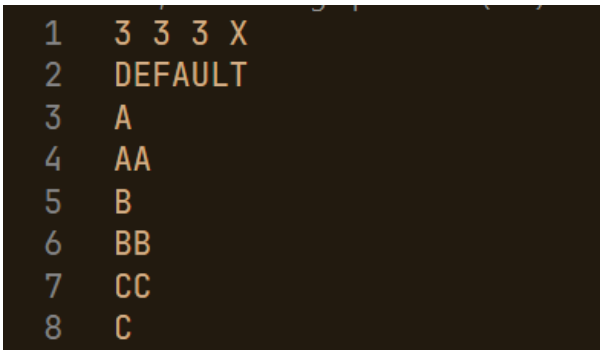
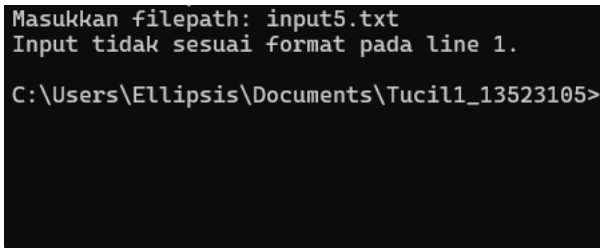
Input	Output
	

Test Case #4 (P Tidak Sesuai)

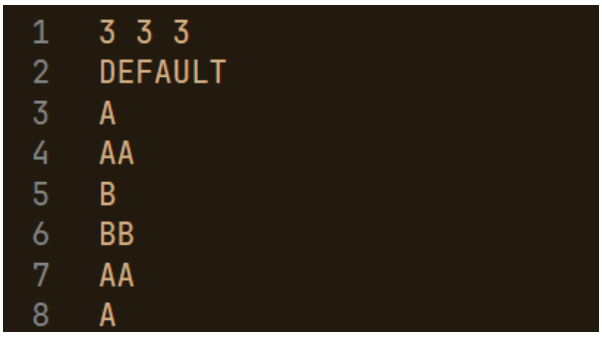
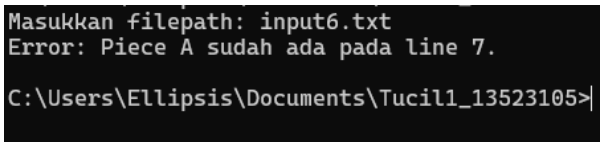
Input	Output

Test Case #5 (Invalid Format)

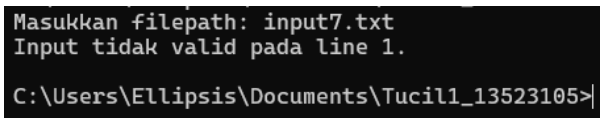
Input	Output
	

Test Case #6 (Appear More Than One)

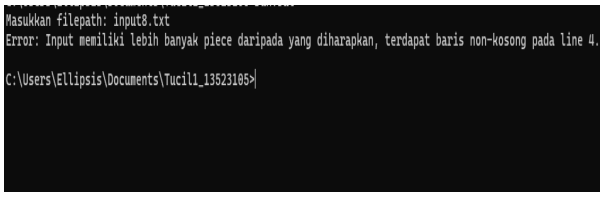
Input	Output
	

Test Case #7 (N or M or P < 1)

Input	Output

Test Case #8 (Piece Size > P)

Input	Output
	

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 serta menyimpan solusi dalam berkas .txt	✓	
5	Program memiliki *Graphical User Interface* (GUI)		✓
6	Program dapat menyimpan solusi dalam bentuk file gambar	✓	
7	Program dapat menyelesaikan kasus konfigurasi *custom*	✓	
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		✓
9	Program dibuat oleh saya sendiri	✓	

Tautan Repositori: https://github.com/fathurwithyou/Tucil1_13523105