

**Tugas Kecil 1 IF2211 Strategi Algoritma**  
**Penyelesaian IQ Puzzle Pro Dengan Algoritma Brute Force**



Disusun Oleh:  
Adiel Rum (10123004)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**TAHUN AKADEMIK 2024/2025**

## Daftar Isi

<b>Overview.....</b>	<b>3</b>
<b>Algoritma dan Kode.....</b>	<b>4</b>
Input.....	4
Pre-processing.....	6
Proses.....	7
Output.....	13
<b>Cara Menjalankan, Test Case, dan Penjelasan.....</b>	<b>15</b>

## Overview



(gambar 1: ilustrasi permainan IQ Puzzle Pro, src:  
<https://www.mytoy.co.za/cdn/shop/products/iib.png?v=1568705703>)

IQ Puzzle Pro adalah permainan puzzle yang menantang pemain untuk menyusun potongan-potongan (pieces) dengan bentuk tertentu ke dalam papan (board) yang telah disediakan. Setiap potongan memiliki bentuk unik, dan tujuan utama permainan ini adalah menempatkan semua potongan ke dalam papan tanpa ada tumpang tindih atau ruang kosong yang tersisa. Permainan ini menguji kemampuan logika, pemecahan masalah, dan visualisasi spasial pemain.

Diberikan beberapa potongan-potongan dengan bentuk dan ukuran tertentu, dan papan dengan ukuran tertentu, dapat digunakan algoritma *brute force* untuk mencari suatu solusi yang mungkin di mana semua potongan memenuhi papan, tidak ada blok di papan yang kosong, dan tidak ada potongan yang tumpang tindih.

Algoritma *brute force* singkatnya adalah algoritma yang menguji semua kemungkinan kasus dari permasalahan (*exhaustive search*) untuk mencari solusi yang diinginkan. Tiap potongan yang dimiliki, akan dicoba untuk diletakkan pada semua posisi yang mungkin pada papan, akan dirotasikan dan dibalik dengan semua cara yang mungkin. Satu-per-satu, potongan akan diletakkan pada papan di posisi dan orientasi tertentu, jika potongan bisa diletakkan pada papan, maka pencarian akan dilanjutkan pada potongan selanjutnya. Jika pada suatu potongan, ternyata sudah tidak ada posisi atau orientasi yang mungkin agar potongannya bisa diletakkan pada papan, maka program akan kembali ke potongan sebelumnya untuk mengganti posisi dan/atau orientasinya, kemudian dilanjutkan kembali ke potongan selanjutnya hingga ditemukan solusi yang diinginkan. Prinsip ini disebut *backtracking*.

Tentu, algoritma *brute force* memiliki keuntungan dan kerugian. Beberapa keuntungannya adalah algoritma brute force cenderung mudah dirumuskan dan diimplementasikan, mengingat algoritma brute force algoritma yang *naive*, dalam kata lain, algoritma brute force adalah algoritma pertama yang muncul pada pikiran. Namun, *brute force* memiliki kelemahan yaitu cenderung tidak efisien, dalam hal waktu maupun *space* untuk memori yang digunakan.

## Algoritma dan Kode

Semua kode yang digunakan terdapat pada file Main.java. Secara umum, kode yang digunakan dibagi menjadi menjadi 4 Bagian: Input, Preprocessing, Process, Output.

### Input

Proses input pada program ini bertujuan untuk membaca data dari file yang berisi informasi tentang papan (board) dan potongan-potongan (pieces) yang akan digunakan dalam permainan IQ Puzzle Pro.

```
static int N, M, P;  
static boolean found = false;  
static int iterationCount = 0;
```

Potongan kode tersebut melakukan deklarasi variabel global, dimana:

- N dan M (int): Dimensi papan
- P (int): Jumlah potongan yang ditempatkan
- Found (boolean): Menandai apakah solusi telah ditemukan (True) atau belum (False)
- iterationCount (int): Menghitung berapa banyak iterasi yang dilakukan

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
  
    System.out.print("Enter the input file name (e.g.,  
input_1.txt): ");  
    String inputFileName = scanner.nextLine();  
  
    String inputFilePath = new File("").getAbsolutePath() +  
"/Tucill1_10123004/test/" + inputFileName;
```

Potongan kode tersebut menginisialisasi scanner yang digunakan, kemudian meminta pengguna untuk memasukkan nama file input. Setelah itu, kode tersebut membangun Path File Input yang akan digunakan

```
try (Scanner fileScanner = new Scanner(new  
File(inputFilePath))) {  
    N = fileScanner.nextInt();  
    M = fileScanner.nextInt();  
    P = fileScanner.nextInt();  
    fileScanner.nextLine();  
  
    String type = fileScanner.nextLine();  
  
    char[][] matrix = new char[N][M];  
  
    if (type.equals("CUSTOM")) {  
        for (int i = 0; i < N; i++) {  
            String line = fileScanner.nextLine().trim();
```

```

        for (int j = 0; j < M; j++) {
            matrix[i][j] = (line.charAt(j) == 'X') ? '*'
: '.';
        }
    } else {
        for (char[] row : matrix) {
            Arrays.fill(row, '*');
        }
    }
}

```

Selanjutnya, nilai-nilai untuk  $N$ ,  $M$ ,  $P$ , dan Jenis Papan dimasukkan, kemudian dilakukan inisialisasi papan. Untuk Jenis Papan “Custom”, papan dimasukkan secara manual, dimana pada koordinat yang mengandung ‘X’, nilai tersebut diubah menjadi ‘\*’ (akan dijelaskan di bagian proses). Sedangkan untuk Jenis Papan “Default”, akan diinisialisasi papan ukuran  $N \times M$  dengan isi semua entrinya adalah ‘\*’.

```

List<List<String>> piecesStrings = new ArrayList<>();
String currentPiece = null;
List<String> currentList = null;

while (fileScanner.hasNextLine()) {
    String line = fileScanner.nextLine();
    if (line.isEmpty()) continue;

    int firstNonWhitespaceIndex = 0;
    while (firstNonWhitespaceIndex < line.length() &&
Character.isWhitespace(line.charAt(firstNonWhitespaceIndex))) {
        firstNonWhitespaceIndex++;
    }

    if (firstNonWhitespaceIndex >= line.length())
continue;

    char firstChar =
line.charAt(firstNonWhitespaceIndex);

    if (currentPiece == null || firstChar !=
currentPiece.charAt(0)) {
        currentPiece = line;
        currentList = new ArrayList<>();
        piecesStrings.add(currentList);
    }
}

```

```
currentList.add(line);  
}
```

Kode memulai proses penginputan potongan dengan menginisialisasi struktur data yang diperlukan. *piecesStrings* adalah sebuah list yang digunakan untuk menyimpan semua potongan, di mana setiap potongan direpresentasikan sebagai list of strings (`List<String>`). Variabel *currentPiece* menyimpan baris pertama dari potongan yang sedang diproses, sementara *currentList* digunakan untuk menyimpan baris-baris yang termasuk dalam potongan tersebut. Struktur data ini memungkinkan kode untuk mengelompokkan baris-baris yang memiliki karakter pertama yang sama ke dalam potongan yang sama.

Selanjutnya, kode membaca setiap baris dari file input. Baris yang kosong atau hanya berisi whitespace akan di-skip. Untuk mengidentifikasi potongan baru, kode mencari karakter pertama yang bukan whitespace pada setiap baris. Jika karakter pertama yang bukan whitespace berbeda dengan karakter pertama dari potongan yang sedang diproses (*currentPiece*), maka baris tersebut dianggap sebagai awal dari potongan baru. Baris-baris yang memiliki karakter pertama yang sama akan dikelompokkan ke dalam potongan yang sama dan disimpan sebagai list of strings dalam *piecesStrings*.

### Pre-processing

Sebelum digunakan untuk proses, data terlebih dahulu diolah menjadi bentuk yang lebih mudah digunakan dalam proses brute force

```
public static List<List<Character>> stringify(List<String> S) {  
    List<List<Character>> result = new ArrayList<>();  
    if (S.isEmpty()) return result;  
  
    int maxLen = 0;  
    for (String s : S) {  
        maxLen = Math.max(maxLen, s.length());  
    }  
  
    for (String s : S) {  
        List<Character> row = new ArrayList<>();  
        for (int i = 0; i < s.length(); i++) {  
            char c = s.charAt(i);  
            if (c == ' ') {  
                row.add('*');  
            } else {  
                row.add(c);  
            }  
        }  
        while (row.size() < maxLen) {  
            row.add('*');  
        }  
    }  
}
```

```

    }
    result.add(row);
}

return result;
}

```

Fungsi *stringify* bertujuan untuk mengubah representasi potongan-potongan dari list of strings (`List<String>`) menjadi list of list of characters (`List<List<Character>>`). Setiap potongan yang awalnya direpresentasikan sebagai baris-baris string akan diubah menjadi matriks karakter, di mana setiap baris string dipecah menjadi karakter-karakter individual. Selain itu, fungsi ini juga memastikan bahwa semua baris dalam sebuah potongan memiliki panjang yang sama dengan menambahkan karakter '\*' sebagai padding jika diperlukan. Hal ini penting karena potongan-potongan dalam IQ Puzzle Pro harus berbentuk persegi panjang agar dapat diproses dengan benar oleh algoritma.

Pada bagian pertama fungsi *stringify*, kode menginisialisasi `result` sebagai list yang akan menyimpan matriks karakter dari potongan. Jika input *S* kosong, fungsi langsung mengembalikan `result` yang kosong. Selanjutnya, kode mencari panjang maksimum dari semua baris dalam potongan (*maxLen*). Panjang ini digunakan untuk memastikan bahwa semua baris dalam potongan memiliki panjang yang sama. Setelah itu, kode memproses setiap baris dalam potongan. Setiap karakter dalam baris diperiksa: jika karakter tersebut adalah spasi (' '), maka akan diganti dengan '\*'; jika bukan, karakter tersebut langsung ditambahkan ke dalam row. Setelah baris selesai diproses, kode menambahkan padding '\*' hingga panjang baris mencapai *maxLen*. Baris yang sudah diproses kemudian ditambahkan ke `result`.

```

List<List<List<Character>>> pieces = new ArrayList<>();
for (List<String> pieceStr : piecesStrings) {
    if (!pieceStr.isEmpty()) {
        pieces.add(stringify(pieceStr));
    }
}

```

Pada kode utama, setelah potongan-potongan dikelompokkan ke dalam *piecesStrings*, kode melakukan konversi setiap potongan dari list of strings menjadi list of list of characters menggunakan fungsi *stringify*. Hasil konversi ini disimpan dalam `pieces`, yang merupakan list of list of list of characters (`List<List<List<Character>>>`). Dengan melakukan konversi ini, setiap potongan siap diproses lebih lanjut oleh algoritma brute force, termasuk untuk dilakukan rotasi, pembalikan, dan penempatan pada papan.

## Proses

Ini adalah inti dari program brute force kali ini!. Tapi, sebelum ke inti dari algoritma brute force-nya, akan dijelaskan fungsi-fungsi yang digunakan

```

public static boolean cekFit(char[][] matrix, List<List<Character>>
piece, int y, int x) {
    int pieceHeight = piece.size();
    if (pieceHeight == 0) return true;
    int pieceWidth = piece.get(0).size();

    if (y + pieceHeight > N || x + pieceWidth > M) {
        return false;
    }

    for (int i = 0; i < pieceHeight; i++) {
        for (int j = 0; j < pieceWidth; j++) {
            if (piece.get(i).get(j) != '*' && matrix[y + i][x +
j] != '*') {
                return false;
            }
        }
    }
    return true;
}

```

Fungsi *cekFit* digunakan untuk memeriksa apakah suatu potongan dapat ditempatkan pada posisi tertentu di papan tanpa menyebabkan tumpang tindih dengan potongan lain atau melampaui batas papan. Fungsi ini menerima papan (matrix), potongan (piece), serta koordinat (y, x) sebagai parameter. Kode akan memeriksa setiap sel pada potongan dan membandingkannya dengan sel yang sesuai di papan. Jika ditemukan sel pada potongan yang bukan '\*' dan sel tersebut di papan juga bukan '\*', maka potongan tidak dapat ditempatkan di posisi tersebut. Fungsi ini mengembalikan true jika potongan dapat ditempatkan dan false jika tidak.

```

public static char[][] place(char[][] original, List<List<Character>>
piece, int y, int x) {
    char[][] newMatrix = new char[N][M];
    for (int i = 0; i < N; i++) {
        System.arraycopy(original[i], 0, newMatrix[i], 0, M);
    }

    int pieceHeight = piece.size();
    if (pieceHeight == 0) return newMatrix;
    int pieceWidth = piece.get(0).size();

    for (int i = 0; i < pieceHeight; i++) {
        for (int j = 0; j < pieceWidth; j++) {

```



```

        if (piece.get(i).get(j) != '*') {
            newMatrix[y + i][x + j] = piece.get(i).get(j);
        }
    }
}
return newMatrix;
}

```

Fungsi *place* berfungsi untuk menempatkan potongan pada papan di posisi yang telah ditentukan. Fungsi ini menerima papan asli (original), potongan (piece), serta koordinat (y, x) sebagai parameter. Kode membuat salinan dari papan asli untuk menghindari modifikasi langsung pada papan utama. Selanjutnya, setiap sel pada potongan yang bukan '\*' akan ditempatkan pada papan salinan di posisi yang sesuai. Fungsi ini mengembalikan papan salinan yang telah diubah, sehingga papan asli tetap utuh dan dapat digunakan untuk percobaan penempatan lainnya.

```

public static List<List<Character>> rotate(List<List<Character>>
piece, String rot) {
    int height = piece.size();
    if (height == 0) return piece;
    int width = piece.get(0).size();

    List<List<Character>> rotated;

    switch (rot) {
        case "0":
            return piece;

        case "90":
            rotated = new ArrayList<>();
            for (int j = 0; j < width; j++) {
                List<Character> newRow = new ArrayList<>();
                for (int i = height - 1; i >= 0; i--) {
                    newRow.add(piece.get(i).get(j));
                }
                rotated.add(newRow);
            }
            return rotated;

        case "180":
            rotated = new ArrayList<>();
            for (int i = height - 1; i >= 0; i--) {
                List<Character> newRow = new ArrayList<>();

```

```

        for (int j = width - 1; j >= 0; j--) {
            newRow.add(piece.get(i).get(j));
        }
        rotated.add(newRow);
    }
    return rotated;

    case "270":
        rotated = new ArrayList<>();
        for (int j = width - 1; j >= 0; j--) {
            List<Character> newRow = new ArrayList<>();
            for (int i = 0; i < height; i++) {
                newRow.add(piece.get(i).get(j));
            }
            rotated.add(newRow);
        }
        return rotated;

    default:
        return piece;
    }
}

```

Fungsi *rotate* digunakan untuk memutar potongan sesuai dengan sudut rotasi yang diberikan. Fungsi ini menerima potongan (*piece*) dan string yang menunjukkan jenis rotasi (*rot*) sebagai parameter. Berdasarkan nilai *rot*, potongan akan diputar 0°, 90°, 180°, atau 270°. Rotasi dilakukan dengan mengubah posisi baris dan kolom pada potongan sehingga menghasilkan orientasi yang baru. Fungsi ini mengembalikan potongan yang telah diputar dalam bentuk list of list of characters.

```

public static List<List<Character>> flip(List<List<Character>> piece,
String flipType) {
    int height = piece.size();
    if (height == 0) return piece;
    int width = piece.get(0).size();

    List<List<Character>> flipped = new ArrayList<>();

    switch (flipType) {
        case "no":
            return piece;

        case "horizontal":

```

```

        for (int i = 0; i < height; i++) {
            List<Character> newRow = new ArrayList<>();
            for (int j = width - 1; j >= 0; j--) {
                newRow.add(piece.get(i).get(j));
            }
            flipped.add(newRow);
        }
        return flipped;

    case "vertical":
        for (int i = height - 1; i >= 0; i--) {
            List<Character> newRow = new
ArrayList<>(piece.get(i));
            flipped.add(newRow);
        }
        return flipped;

    case "both":
        for (int i = height - 1; i >= 0; i--) {
            List<Character> newRow = new ArrayList<>();
            for (int j = width - 1; j >= 0; j--) {
                newRow.add(piece.get(i).get(j));
            }
            flipped.add(newRow);
        }
        return flipped;

    default:
        return piece;
    }
}

```

Fungsi *flip* bertujuan untuk membalik potongan secara horizontal, vertikal, atau keduanya. Fungsi ini menerima potongan (*piece*) dan string yang menunjukkan jenis pembalikan (*flipType*) sebagai parameter. Berdasarkan nilai *flipType*, potongan akan dibalik secara horizontal, vertikal, atau kedua-duanya. Pembalikan dilakukan dengan mengubah urutan baris atau kolom pada potongan. Fungsi ini mengembalikan potongan yang telah dibalik dalam bentuk list of list of characters.

```

public static boolean isBoardComplete(char[][] matrix) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            if (matrix[i][j] == '*') {

```

```

        return false;
    }
}
return true;
}

```

Fungsi *isBoardComplete* digunakan untuk memeriksa apakah papan telah terisi sepenuhnya tanpa ada ruang kosong (\*). Fungsi ini menerima papan (matrix) sebagai parameter dan memeriksa setiap sel pada papan. Jika ditemukan sel yang masih berisi '\*', fungsi akan mengembalikan false, yang menandakan bahwa papan belum terisi sepenuhnya. Jika semua sel terisi, fungsi mengembalikan true, yang menandakan bahwa papan telah terisi dengan benar dan solusi telah ditemukan.

```

public static void bruteforce(char[][] matrix,
List<List<List<Character>>> pieces, int idx) {
    if (found || idx >= pieces.size()) return;

    if (isBoardComplete(matrix)) {
        display(matrix);
        System.out.println();
        found = true;
        return;
    }

    String[] rotations = {"0", "90", "180", "270"};
    String[] flips = {"no", "horizontal", "vertical", "both"};

    for (String rot : rotations) {
        for (String flipType : flips) {
            List<List<Character>> transformedPiece =
rotate(pieces.get(idx), rot);
            transformedPiece = flip(transformedPiece, flipType);

            for (int y = 0; y < N; y++) {
                for (int x = 0; x < M; x++) {
                    if (cekFit(matrix, transformedPiece, y, x)) {
                        iterationCount++; // Menghitung setiap
iterasi

                        char[][] newMatrix = place(matrix,
transformedPiece, y, x);
                        bruteforce(newMatrix, pieces, idx + 1);
                        if (found) return;
                    }
                }
            }
        }
    }
}

```

```

    }
    }
    }
    }
    }
}

```

Fungsi *bruteforce* adalah inti dari algoritma pencarian solusi. Fungsi ini menerima papan (matrix), list potongan (pieces), dan indeks potongan yang sedang diproses (idx) sebagai parameter. Kode akan mencoba semua kombinasi rotasi dan pembalikan untuk setiap potongan, lalu menempatkannya pada setiap posisi yang mungkin di papan. Jika potongan dapat ditempatkan, fungsi akan memanggil dirinya sendiri secara rekursif untuk memproses potongan berikutnya. Jika semua potongan berhasil ditempatkan dan papan terisi sepenuhnya, solusi akan ditampilkan. Jika tidak, fungsi akan kembali ke potongan sebelumnya untuk mencoba kombinasi lain (backtracking).

## Output

Jika solusi ditemukan (found = true), program akan menampilkan papan yang telah terisi dengan potongan-potongan yang valid. Jika tidak ada solusi yang ditemukan, program akan menampilkan pesan "No solution found!". Selain itu, program juga menampilkan informasi tambahan seperti jumlah iterasi yang dilakukan dan waktu eksekusi program.

```

public static void display(char[][] matrix) {
    Map<Character, String> colorMap = new HashMap<>();
    colorMap.put('A', "\u001B[31m");
    colorMap.put('B', "\u001B[32m");
    colorMap.put('C', "\u001B[33m");
    colorMap.put('D', "\u001B[34m");
    colorMap.put('E', "\u001B[35m");
    colorMap.put('F', "\u001B[36m");
    colorMap.put('G', "\u001B[37m");
    colorMap.put('H', "\u001B[91m");
    colorMap.put('I', "\u001B[92m");
    colorMap.put('J', "\u001B[93m");
    colorMap.put('K', "\u001B[94m");
    colorMap.put('L', "\u001B[95m");
    colorMap.put('M', "\u001B[96m");
    colorMap.put('N', "\u001B[97m");
    colorMap.put('O', "\u001B[41m");
    colorMap.put('P', "\u001B[42m");
    colorMap.put('Q', "\u001B[43m");
    colorMap.put('R', "\u001B[44m");
    colorMap.put('S', "\u001B[45m");
}

```

```

        colorMap.put('T', "\u001B[46m");
        colorMap.put('U', "\u001B[47m");
        colorMap.put('V', "\u001B[101m");
        colorMap.put('W', "\u001B[102m");
        colorMap.put('X', "\u001B[103m");
        colorMap.put('Y', "\u001B[104m");
        colorMap.put('Z', "\u001B[105m");

        String resetColor = "\u001B[0m";

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                char c = matrix[i][j];
                if (c == '.') {
                    System.out.print(" ");
                } else {
                    String color = colorMap.getOrDefault(c,
resetColor);

                    System.out.print(color + c + resetColor);
                }
            }
            System.out.println();
        }
    }
}

```

```

try (PrintWriter writer = new PrintWriter(new
FileWriter(outputFilePath))) {
    if (found) {
        writer.println("Solution found:");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                writer.print(matrix[i][j] + " ");
            }
            writer.println();
        }
    } else {
        writer.println("No solution found!");
    }
    writer.println("Total iterations: " +
iterationCount);
    writer.println("Execution time: " + executionTime

```

```
+ " ms");  
  
        System.out.println("Output saved to " +  
outputFilePath);
```

## Cara Menjalankan, Test Case, dan Penjelasan

### Setup

1. Clone repository [https://github.com/adielrum/Tucil1\\_10123004](https://github.com/adielrum/Tucil1_10123004)
2. Pindahkan directory ke lokasi file dengan cd/src
3. Jalankan program dengan perintah “java main”

### Format Input

N, M, P

Tipe\_papan

<piece\_1>

<piece\_2>

...

<piece\_P>

Catatan: program sedikit ngebug jika baris pertama dari piece, tidak diawali dengan char, contoh:

A

AA

Untuk menanggulangi, diusahakan baris pertama dari piece diawali dengan char

Contoh:

AA	atau	A
A		AA

### Format Output

a). Jika ditemukan solusi

XXX ... X

XX.

X..

...

XXX ... X

Total iterations:

Execution time:


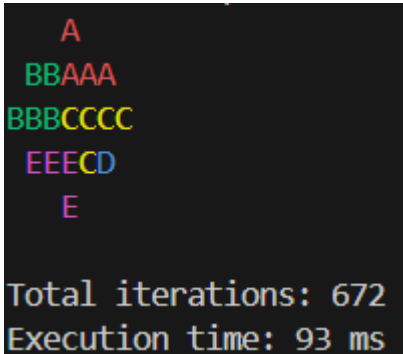
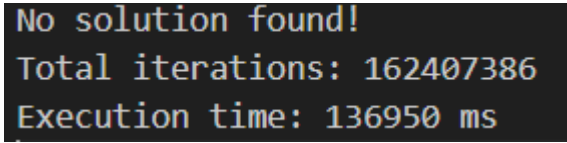
b). Jika tidak ditemukan solusi

No solution found!

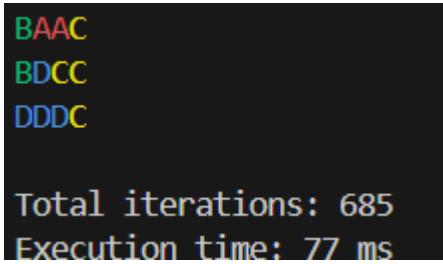
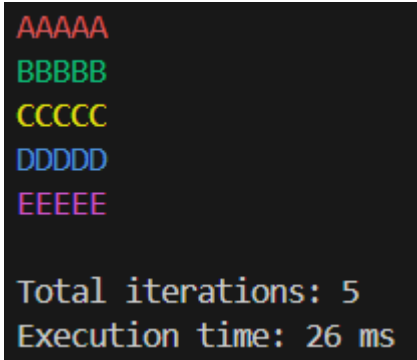
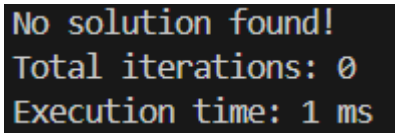
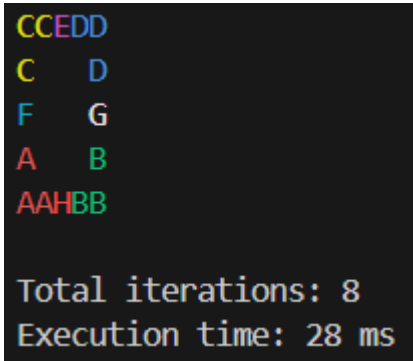
Total iterations:

Execution time:

### Contoh Test Case:

No.	Test Case	Output
1.	5 5 8 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	 <pre> AGGGC AABCC EEBBF EEDFF EDDFF  Total iterations: 118374 Execution time: 1331 ms           </pre>
2.	5 7 5 CUSTOM ...X... .XXXXX. XXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E	 <pre> A BBAAA BBBCCCC EEECD E  Total iterations: 672 Execution time: 93 ms           </pre>
3.	4 4 4 DEFAULT A AA B BB C CC D	 <pre> No solution found! Total iterations: 162407386 Execution time: 136950 ms           </pre>



	DD	
4.	3 4 4 DEFAULT AA B B C CC C D DD D	
5.	5 5 5 DEFAULT AAAAA B B B B B CCCCC D D D D D EEEE	
6.	5 1 3 DEFAULT A AA B BB C C C	
7.	5 5 8 CUSTOM XXXXX X...X X...X X...X XXXXX A AA B	

	BB C CC D DD E F G H	
--	--	--

**Tabel Checklist**

No.	Poin	Ya	Tidak
1.	Program berhasil dikompilasi tanpa kesalahan	<input checked="" type="checkbox"/>	
2.	Program berhasil dijalankan	<input checked="" type="checkbox"/>	
3.	Solusi yang diberikan program benar dan mematuhi aturan permainan	<input checked="" type="checkbox"/>	
4.	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	<input checked="" type="checkbox"/>	
5.	Program memiliki Graphical User Interface (GUI)		<input checked="" type="checkbox"/>
6.	Program dapat menyimpan solusi dalam bentuk file gambar		<input checked="" type="checkbox"/>
7.	Program dapat menyelesaikan kasus konfigurasi custom	<input checked="" type="checkbox"/>	
8.	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		<input checked="" type="checkbox"/>
9.	Program dibuat oleh saya sendiri	<input checked="" type="checkbox"/>	