Tugas Kecil I IF2211 Strategi Algoritma

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Disusun oleh:

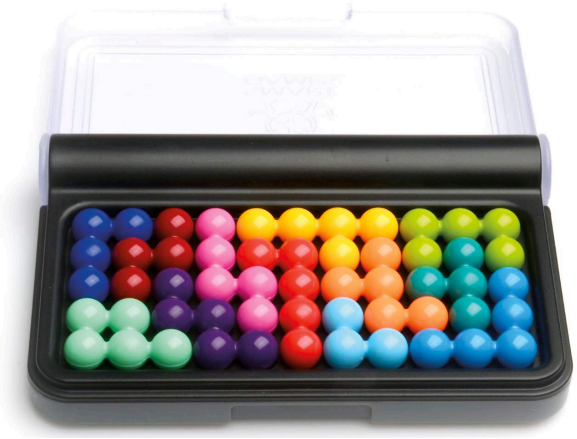
Timothy Niels Ruslim (10123053)

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

Daftar Isi

Overview

Akan diselesaikan masalah mencari solusi dari suatu puzzle berisi balok serupa tetromino yang sering terlihat pada permainan IQ Puzzler Pro (Gambar 1). Inti game adalah untuk memenuhi papan puzzle secara penuh dengan balok-balok serupa tetromino yang berwarna berbeda. Balok-balok dapat dirotasikan maupun direfleksikan sebelum diletakkan pada papan. Ini menghasilkan suatu puzzle yang menyulitkan.



Gambar 1. IQ Puzzler Pro

Untuk menyelesaikan masalah ini, akan digunakan algoritma brute-force murni tanpa heuristik. Algoritma brute-force atau algoritma exhaustive bekerja dengan mencari semua kombinasi atau kandidat solusi, lalu menguji apakah solusi tersebut benar. Dalam kasus permainan ini, dicari semua kombinasi balok-balok yang muat dalam papan, lalu dicek apakah kombinasi tersebut memenuhi seluruh papan.

Saat diterapkan, algoritma ini juga menggunakan rekursi dan backtracking. Untuk mencari suatu kandidat solusi, diletakkan terlebih dahulu suatu balok di papan. Lalu, sekarang masalah menjadi papan yang beberapa selnya telah dihilangkan (digunakan balok pertama) dengan himpunan balok-balok yang lebih sedikit. Maka, rekursi menjadi solusi baik untuk melakukan ini. Lalu, misalkan algoritma terus menelusuri solusi papan dengan menaruh balok-balok selanjutnya pada lokasi-lokasi lainnya secara acak. Jika brute-force tersebut ternyata menghasilkan suatu kasus yang tidak menyelesaikan papan, itu berarti pemilihan permutasi dan lokasi balok pertama tersebut itu salah. Jadi, algoritma akan kembali ke balok pertama dan mencoba merotasikannya, merefleksikannya, atau menaruhnya pada lokasi lain. Itulah backtracking.

Adapun bahwa algoritma yang digunakan melakukan early pruning dengan menghitung apakah jumlah sel pada papan sudah sama dengan jumlah kotak balok. Perhatikan bahwa ini bukan heuristik, karena tidak diintroduksikan suatu fungsi bias yang menggerakkan algoritma pada solusi, melainkan hanya mencegah komputasi yang tidak diperlukan.

Program

Algoritma penyelesaian permainan IQ Puzzler Pro ini diimplementasikan di bahasa pemrograman Java. Cara menggunakan dan membaca hasil program dapat dilihat pada `README` di repository.

Input

Seperti pada petunjuk tucil, input berupa file `.txt` yang memiliki format bersesuaian. Saat program dijalankan, user dapat menulis alamat file tersebut, atau hanya nama filenya (dengan extension) jika sudah berada di folder `test`. Berikut implementasinya menggunakan package `Scanner` dan `File`.

```
Scanner terminalScanner = new Scanner(System.in);
File file = null;
boolean fileExists = false;
while (!fileExists) {

    System.out.print("Masukkan alamat file: ");
    String path = terminalScanner.nextLine( );
    File option1 = new File(path);
    File option2 = new File("../test/" + path);

    if (option1.exists()) {
        file = option1;
        fileExists = true;
    } else if (option2.exists()) {
        file = option2;
        fileExists = true;
    } else {
        System.out.print("File tidak ditemukan. Coba lagi. ");
    }

}
```

Berikutnya, jika file sudah valid, kita akan membaca informasi di dalamnya. Informasi `m`, `n`, dan `p` ditaruh dalam array dimensions, juga tipe permainan pada variabel `type`. Kemudian, menggunakan informasi tersebut, dibuat sebuah papan permainan dengan objek `Board`. Ada juga objek `Block` yaitu balok yang akan ditaruh. Penjelasan objek-objek tersebut akan dijelaskan pada subbab berikutnya. Dengan setiap baris yang dibaca, kita menaruhnya pada suatu `Block`, lalu jika huruf yang terkandung pada baris berikutnya sudah berbeda (`block.id`), akan dibuat `Block` yang baru untuk mengulang proses ini. Perhatikan juga bahwa untuk setiap balok, dicari juga semua permutasi (rotasi dan refleksinya) secara langsung, dan dibuat menjadi atribut `Block` tersebut secara langsung.

```
Scanner fileScanner = new Scanner(file);

// Create Board
```

```
int[] dimensions = toInt(fileScanner.nextLine().split(" "));
String type = fileScanner.nextLine();
Board board = new Board(dimensions[0], dimensions[1], dimensions[2], type,
fileScanner);

// Get the Pieces
ArrayList<Block> blocks = new ArrayList<>();
Block block = null;

while (fileScanner.hasNextLine()) {
    char[] row = fileScanner.nextLine().toCharArray();
    char letter = findId(row);

    if (block == null || block.id != letter) {
        if (block != null) {
            block.permutations = block.permuteBlock();
            blocks.add(block);
        }
        block = new Block(letter);
    }

    block.addRow(row);
}

// Last Row
if (block != null) {
    block.permutations = block.permuteBlock();
    blocks.add(block);
}

fileScanner.close();
```

Block

Dibuat objek `Block` yaitu hanyalah `ArrayList` dua dimensi berisi karakter, atau dalam kata lain matriks, untuk merepresentasikan balok. Dia memiliki atribut sebagai berikut.

Atribut	Tipe Data	Deskripsi
`id`	`char`	Huruf yang direpresentasikan balok, digunakan pada input seperti di atas, tetapi juga nanti saat perlu menaruh atau mengeluarkan balok dari papan.
`num`	`int`	Jumlah huruf pada balok, digunakan pada pruning awal.
`permutations`	`LinkedHashSet <Block>`	List semua permutasi (rotasi/refleksi) dari balok, yang digunakan saat brute-force. Permutasi menjadi atribut agar menghemat waktu penyelesaian algoritma nanti (tidak perlu mencari permutasi dalam loop).

Tidak dibuat atribut private agar tidak perlu getter dan setter untuk memudahkan program (karena difokuskan pada algoritma). Ini berlaku untuk objek lainnya.

```
public class Block extends ArrayList<ArrayList<Character>> {

    char id;
    int num = 0;
    LinkedHashSet<Block> permutations = new LinkedHashSet<>();

    public Block(char id) {
        super();
        this.id = id;
    }

    ...

}
```

Selanjutnya, ada juga beberapa method dari balok yang akan membantu.

Method	Parameter	Return	Deskripsi
`addRow`	`char[]`	-	Menambah baris pada balok, digunakan pada input di atas.
`displayBlock`	-	-	Hanya untuk print balok pada terminal.
`rotateBlock`	-	`Block`	Merotasikan balok 90 derajat ke arah jarum jam menggunakan re-indexing pada for loop.
`reflectBlock`	-	`Block`	Merefleksikan balok terhadap sumbu vertikal menggunakan re-indexing pada for loop.
`permuteBlock`	-	`LinkedHashSet <Block>`	Menggunakan fungsi rotasi dan refleksi di atas untuk mencari semua permutasi. Karena menggunakan HashSet, tidak akan ada duplikat.
`padBlock`	-	`Block`	Ditaruh karakter titik (·) pada sel-sel matriks yang tidak ada huruf.
`width`	-	`int`	Mencari baris yang terpanjang sehingga menjadi patokan untuk lebar balok saat perlu `padBlock`.

Berikut kode untuk fungsi-fungsi di atas.

```
public void addRow(char[] row) {
    ArrayList<Character> rowList = new ArrayList<>();
    for (char letter : row) {
        rowList.add(letter);
        if (letter == this.id) num++;
    }
    this.add(rowList);
    this.padBlock();
}

public void displayBlock() {
    for (ArrayList<Character> row : this) {
        for (Character letter : row) {
            System.out.print(letter + " ");
        }
        System.out.println();
    }
}

public Block rotateBlock() { // rotate 90 degrees clockwise
    Block rotatedBlock = new Block(this.id);
    int rows = width();
    int cols = this.size();

    for (int i = 0; i < rows; i++) {
        rotatedBlock.add(new ArrayList<>());
    }
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++) {
            char letter = this.get(i).get(j);
            rotatedBlock.get(j).add(0, letter);
        }
    }

    return rotatedBlock;
}

public Block reflectBlock() { // reflect about y-axis
    Block reflectedBlock = new Block(this.id);
    int rows = this.size();

    for (int i = 0; i < rows; i++) {
        reflectedBlock.add(new ArrayList<>());
        for (char letter : this.get(i)) {
            reflectedBlock.get(i).add(0, letter);
        }
    }

    return reflectedBlock;
}

// get all permutations (by rotation and reflection)
public LinkedHashSet<Block> permuteBlock() {
    LinkedHashSet<Block> permutations = new
LinkedHashSet<>(Arrays.asList(this));

    Block block = this;
```

```
    for (int i = 0; i < 4; i++) {
        block = block.rotateBlock();
        permutations.add(block);
        permutations.add(block.reflectBlock());
    }

    return permutations;
}

// add dots in empty spaces
public Block padBlock() {
    Block paddedBlock = new Block(this.id);
    int maxWidth = width();
    for (ArrayList<Character> row : this) {
        for (int i = 0; i < row.size(); i++) {
            if (row.get(i) == ' ') row.set(i, '·');
        }
        while (row.size() < maxWidth) {
            row.add('·');
        }
    }
    return paddedBlock;
}

// get width before it is padded
public int width() {
    int maxWidth = 0;
    for (ArrayList<Character> row : this) {
        maxWidth = Math.max(maxWidth, row.size());
    }
    return maxWidth;
}
```

Board

Dibuat objek `Board` yaitu hanyalah `ArrayList` dua dimensi berisi karakter, atau dalam kata lain matriks, untuk merepresentasikan papan permainan. Dia memiliki atribut sebagai berikut.

Atribut	Tipe Data	Deskripsi
`n`	`int`	Jumlah baris dari papan.
`m`	`int`	Jumlah kolom dari papan.
`p`	`int`	Jumlah balok yang ingin ditaruh pada papan.
`cells`	`int`	Jumlah sel yang dapat diisi balok. Secara default, ini adalah m × n, tetapi untuk konfigurasi tipe custom, dia akan dihitung berdasarkan jumlah X pada file.
`type`	`String`	Tipe papan antara default, custom, atau

		pyramid.
`cases`	`int`	Jumlah kasus yang telah dicoba pada papan oleh algoritma.

Adapun bahwa konstruktor secara default menghasilkan matriks berisi karakter titik (·) berdimensi `m` × `n`. Tetapi, jika tipe puzzle custom, program hanya mengisi titik pada sel-sel yang berkorespondensi dengan `X` di file input. Ini mengapa suatu `Scanner` menjadi parameter dari konstruktor papan.

```
public class Board extends ArrayList<ArrayList<Character>> {

    // Board properties
    int n, m, p, cells;
    String type;

    // Algorithm stuff
    int cases = 0;

    // Constructor
    public Board(int n, int m, int p, String type, Scanner fileScanner) {
        super();
        this.n = n;
        this.m = m;
        this.p = p;
        this.type = type;
        cells = n * m;

        switch(type) {
            case "DEFAULT":
                for (int i = 0; i < n; i++) {
                    ArrayList<Character> row = new ArrayList<>();
                    for (int j = 0; j < m; j++) {
                        row.add('·');
                    }
                    this.add(row);
                }
                break;
            case "CUSTOM":
                cells = 0;
                for (int i = 0; i < n; i++) {
                    char[] input = fileScanner.nextLine().toCharArray();
                    ArrayList<Character> row = new ArrayList<>();
                    for (int j = 0; j < m; j++) {
                        if (input[j] == 'X') {
                            row.add('·');
                            cells++;
                        } else {
                            row.add(' ');
                        }
                    }
                    this.add(row);
                }
                break;
            case "PYRAMID":
                System.out.println("Too difficult...");
```

```
        }
    }
    ...
}
```

Adapun beberapa method dari objek papan ini seperti berikut.

Method	Parameter	Return	Deskripsi
`displayBoard`	-	-	Hanya untuk print papan. Tetapi, dalam kasus papan sudah ada balok di dalamnya, akan juga diwarnakan. Warna menggunakan kode ANSI dengan cara mengambil `p` warna secara serata mungkin dari 26 warna pelangi yang sudah dibuat dari roda RGB (format ANSI untuk RGB adalah `\u001B[38;2;%R;%G;%Bm`).
`canPlaceBlock`	-	`boolean`	Mengecek apakah suatu balok dapat diletakkan pada papan secara valid (tidak keluar papan atau tumpang tindih), secara spesifik jika sudut atas kiri dari balok dapat diletakkan pada suatu sel (`i,j`) pada papan.
`placeBlock`	`Block, int, int`	`Board`	Letakkan balok pada papan di sel (`i,j`). Ini mudah dilakukan dengan loop relatif balok pada papan.
`removeBoard`	`Block, int, int`	`Board`	Hilangkan balok pada papan di sel (`i,j`). Ini mudah dilakukan dengan loop relatif balok pada papan.
`solveBoard`	`ArrayList <Block>`	`Boolean`	Inilah **jantung** dari programnya! Fungsi ini yang mengimplementasikan algoritma brute-force dengan backtracking dan early pruning berdasarkan rekursi. Penjelasan lebih rinci ada di subbab berikutnya.

Berikut source code untuk fungsi-fungsi di atas.

```
public void displayBoard() {

    // Possible 26 Colors
    String RESET = "\u001B[0m";
    String[] rainbow = {
        "\u001B[38;2;255;0;0m",
        "\u001B[38;2;255;59;0m",
        "\u001B[38;2;255;118;0m",
        "\u001B[38;2;255;177;0m",
        "\u001B[38;2;255;235;0m",
        "\u001B[38;2;216;255;0m",
        "\u001B[38;2;157;255;0m",
        "\u001B[38;2;98;255;0m",
        "\u001B[38;2;39;255;0m",
        "\u001B[38;2;0;255;20m",
        "\u001B[38;2;0;255;78m",
        "\u001B[38;2;0;255;137m",
        "\u001B[38;2;0;255;196m",
        "\u001B[38;2;0;255;255m",
        "\u001B[38;2;0;196;255m",
        "\u001B[38;2;0;137;255m",
        "\u001B[38;2;0;78;255m",
        "\u001B[38;2;0;20;255m",
        "\u001B[38;2;39;0;255m",
        "\u001B[38;2;98;0;255m",
        "\u001B[38;2;157;0;255m",
        "\u001B[38;2;216;0;255m",
        "\u001B[38;2;255;0;235m",
        "\u001B[38;2;255;0;177m",
        "\u001B[38;2;255;0;118m",
        "\u001B[38;2;255;0;59m"
    };

    // Printing the Board
    Map<Character, Integer> letterColors = new HashMap<>();
    for (ArrayList<Character> row : this) {
        for (char letter : row) {
            if (letter != '·' && letter != ' ') {
                // map letter to color
                if (!letterColors.containsKey(letter)) {
                    letterColors.put(letter, (int) letterColors.size() *
25/p);
                }
                System.out.print(rainbow[letterColors.get(letter)] + letter
+ RESET + " ");
            } else {
                System.out.print(letter + " ");
            }
        }
        System.out.println();
    }

}

public boolean canPlaceBlock(Block block, int x, int y) {
    int row = block.size();
    int col = block.get(0).size();
```

```
    if (x + row > n || y + col > m) {
        return false;
    }
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            if (block.get(i).get(j) != '·' && this.get(i + x).get(j + y) !=
'·') {
                return false;
            }
        }
    }
    return true;
}

public Board placeBlock(Block block, int x, int y) {
    Board newBoard = this;
    int row = block.size();
    int col = block.get(0).size();

    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            if (block.get(i).get(j) != '·') {
                newBoard.get(i + x).set(j + y, block.id);
            }
        }
    }
    return newBoard;
}

public Board removeBlock(Block block, int x, int y) {
    Board newBoard = this;
    int row = block.size();
    int col = block.get(0).size();

    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            if (block.get(i).get(j) == block.id) {
                newBoard.get(i + x).set(j + y, '·');
            }
        }
    }
    return newBoard;
}
```

Algoritma

Seperti yang telah dijelaskan, fungsi utama untuk algoritma brute-force adalah suatu method dari objek `Board`. Ini dilakukan supaya semua operasi algoritma dilakukan secara langsung terhadap papan. Sebelum menjelaskan algoritmanya, sebaiknya melihat terlebih dahulu source code-nya secara langsung.

```
public boolean solveBoard(ArrayList<Block> blocks) {

    // Failed if more pieces than spaces
```

```
    if (cases == 0) {
        int num = 0;
        for (Block block : blocks) num += block.num;
        if (num != cells) return false;
    }

    // Solved if every piece is placed
    if (blocks.isEmpty()) return true;

    // Try every position
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {

            // Try every piece
            for (int k = 0; k < blocks.size(); k++) {
                Block block = blocks.get(k);

                // Try Every permutation
                for (Block attemptedBlock : block.permutations) {
                    if (canPlaceBlock(attemptedBlock, i, j)) {

                        // Recursion
                        cases++;

                        Block successfulBlock = blocks.remove(k);
                        placeBlock(attemptedBlock, i, j);

                        if (solveBoard(blocks)) return true;

                        removeBlock(attemptedBlock, i, j);
                        blocks.add(k, successfulBlock);

                    }
                }

            }

        }
    }

    return false;

}
```

Pertama, fungsi ini menerima parameter suatu list berisi semua balok-balok yang harus diisi pada papan. Hal pertama yang dicek adalah apakah jumlah kotak pada semua balok (yang dihitung dengan menjumlahkan semua atribut `num` balok) sama dengan jumlah sel kosong pada papan (yaitu atribut `cells` papan). Ini berarti jumlah kotak yang lebih kecil maupun lebih besar akan tetap menghasilkan output “Tidak ada solusi.” Perhatikan bahwa “pruning” ini hanya dilakukan pada iterasi pertama, karena tentu tidak perlu menghitung terus apa yang sudah diketahui tentang papan dan balok berkali-kali. Lalu, kita mengetahui bahwa program selesai saat semua balok yang dimiliki telah diletakkan pada papun. Pendekatan ini lebih hemat waktu daripada mengecek apakah masih ada sel yang kosong pada papan.

Kemudian, untuk brute force-nya sendiri, algoritma akan mengiterasi seluruh sel pada matriks. Lalu, akan dicoba dari list balok-balok suatu balok untuk diletakkan pada sel. Jika mungkin ditaruh (valid berdasarkan fungsi yang telah dibuat), maka balok akan diletakkan, lalu fungsi akan melakukan rekursi dengan papan yang baru dan list yang diperpendek. Jika gagal, akan dicoba permutasi lain dari balok tersebut. Jika semua permutasi gagal, akan dilanjutkan ke balok lainnya. Jika semua balok dan permutasinya gagal, akan dilanjutkan ke sel berikutnya pada matriks. Itulah urutan brute-force. Perhatikan juga penggunaan output boolean yang ideal untuk backtracking.

Beberapa optimasi yang dilakukan pada algoritma adalah sebagai berikut.

- Menghitung apakah jumlah kotak balok dan sel kosong papan sama sebagai early pruning.
- Menggunakan `blocks.isEmpty()` daripada mengecek apakah ada sel kosong pada papan (mengurangi penggunaan loop berlebihan).
- Menggunakan `blocks.get(k)` terlebih dahulu, dan hanya `remove` saat memang balok dapat diletakkan (karena peng-update-an array mengambil $O(k)$ waktu).
- Permutasi yang telah menjadi atribut balok, sehingga tidak perlu dihitung terus setiap iterasi atau rekursi.

Pada fungsi `main` di `App.java`, fungsi ini hanya perlu digunakan sebagai berikut.

```
long start = System.nanoTime();
if (board.solveBoard(blocks)) {
    board.displayBoard();
} else {
    System.out.println("Tidak ada solusi.");
}
long end = System.nanoTime();
```

Komentar. Algoritma dapat dibuat lebih cepat dengan memoization. Yaitu, algoritma akan menghafal kasus yang telah ditelusuri dan hanya menelusuri suatu kasus jika belum pernah menelusurinya. Ini dapat dilakukan dengan membuat atribut `HashSet<String> failedStates` pada papan. Lalu, pada algoritma brute-force, dapat di-update

`failedStates.add(this.toString())`,

setiap kali suatu kasus gagal. Lalu, sebelum setiap iterasi, dapat dicek

`if (failedStates.contains(this.toString())) return false`,

sehingga kasus tidak akan dicoba oleh algoritma jika sudah pernah dilakukan. Penerapan memoization ini dapat memangkas setengah waktu dari waktu penyelesaian. Misalnya, contoh 1 yang ada pada spesifikasi Tucil 1 ini dapat diselesaikan dalam hanya `59 ms`. Akan tetapi, walaupun pendekatan ini lebih cepat, dia memakan memory yang sangat besar karena semua kemungkinan letak balok pada papan sangat amat banyak. Untuk papan yang cukup besar (dengan sel lebih banyak dari ~50), `failedStates` akan mengalami overflow.

Output

Hasil dari program antara papan dengan baloknya (dengan sudah ada warna) jika ada solusi, atau tulisan "Tidak ada solusi" jika tidak ada. Tetapi, dicatat juga waktu penyelesaian algoritma (dengan `System.nanoTime()`) dan banyak kasus yang ditelusuri brute force (dengan `board.cases`). Jika diinginkan, user juga dapat menyimpan solusi tersebut (yang mengandung solusi papan, waktu penyelesaian, dan kasus yang ditelusuri) dalam suatu file `.txt` yang secara otomatis ditaruh pada folder `test`, dengan nama yang sama dengan file input tetapi dengan suffiks `_Solution.txt`.

```
System.out.println("\nWaktu pencarian: " + (end - start) / 1000000 + "
ms");
System.out.println("\nBanyak kasus yang ditinjau: " + board.cases);

// Write Output into File
System.out.print("\nApakah anda ingin menyimpan solusi? (ya/tidak) ");
String answer = terminalScanner.nextLine();
if (answer.equals("ya")) {
    File solution = new File("../test/" + getName(file) + "_Solution.txt");
    PrintWriter solutionWriter = new PrintWriter(new FileWriter(solution));
    for (ArrayList<Character> r : board) {
        for (char c : r) {
            solutionWriter.print(c + " ");
        }
        solutionWriter.println();
    }
    solutionWriter.println("\nWaktu pencarian: " + (end - start) / 1000000
+ " ms");
    solutionWriter.print("\nBanyak kasus yang ditinjau: " + board.cases);
    solutionWriter.close();
}

// End Program
System.out.println();
terminalScanner.close();
```

Contoh

Berikut beberapa contoh input pada program penyelesaian puzzle.

1. Contoh 5 × 5 Default Solvable (Spesifikasi)

```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

```
Masukkan alamat file: Problem_1.txt

A B B C C
A A B C D
E E E D D
E E F F F
G G G F F

Waktu pencarian: 74 ms

Banyak kasus yang ditinjau: 17111

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

2. Contoh 6 × 6 Default Solvable

```
6 6 9
DEFAULT
AAAA
B
BB
 B
CCC
C
DD
DD
EE
 EE
F
FF
F
GG
GG
 H
HH
 H
IIII
```

```
Masukkan alamat file: Problem_2.txt

A A A A C I
B B C C C I
F B B D D I
F F E D D I
F H E E G G
H H H E G G

Waktu pencarian: 110 ms

Banyak kasus yang ditinjau: 23276

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

3. Contoh 4 × 4 Default Solvable

```
4 4 4
DEFAULT
A
AA
BB
B
B
CC
 CC
D
D
DDD
```

```
A A C C
A C C D
B B B D
B D D D

Waktu pencarian: 12 ms

Banyak kasus yang ditinjau: 179

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

4. Contoh 8 × 8 Default Not Solvable

```
8 8 13
DEFAULT
AAAAA
BB
B
BB
CCCC
   C
  D
DDDD
 E
 E
EEE
FFF
FF
GGG
GG
 H
HHH
 H
 I
 I
II
I
  J
 JJ
JJ
KK
 K
 KK
L
LLL
 L
MMM
  M
  M
```

```
Masukkan alamat file: Problem_4.txt

Tidak ada solusi.

Waktu pencarian: 0 ms

Banyak kasus yang ditinjau: 0

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

5. Contoh 5 × 11 Default Solvable

```
5 11 12
DEFAULT
AA
A
A
BB
 BB
C
C
CC
 C
DDDD
  D
FF
 F
 F
 F
EEE
E E
 G
GG
 GG
 H
HH
 H
L
LL
 LL
III
  I
  I
J
JJ
KK
KKK
```

```
A A B C F H H H I I I
A B B C F L H E E E I
A B C C F L L E G E I
K K C F F D L L G G J
K K K D D D D G G J J

Waktu pencarian: 8934300 ms

Banyak kasus yang ditinjau: 1621747920

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

6. Contoh 4 × 8 Default Solvable

```
4 8 7
DEFAULT
AAAA
A
BBBB
  B
  CC
CCCC
DD
D
 E
EEEE
```

```
Masukkan alamat file: Problem_6.txt

A A A A B B B B
A C C C C B D D
F F E C C G D G
F E E E E G G G

Waktu pencarian: 7224 ms

Banyak kasus yang ditinjau: 3285725

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

```
FF
 F
G G
GGG
```

7. Contoh 7 × 5 Default Solvable

```
7 5 8
DEFAULT
A
A
AA
A
BBBB
   B
C
C
CC
DD
DDDD
EE
EE
E
FF
 F
GGGG
H
HH
```

```
Masukkan alamat file: Problem_7.txt

A B B B B
A C C C B
A A G C E
A F G E E
F F G E E
D D G H H
D D D D H

Waktu pencarian: 206 ms

Banyak kasus yang ditinjau: 40465

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

8. Contoh 5 × 7 Default Solvable

```
5 7 7
DEFAULT
A
A
AA
 A
BBB
BB
 C
 C
CCC
  D
DDDD
E
EEE
  E
FF
F
FF
 G
GG
```

```
Masukkan alamat file: Problem_8.txt

A B B C C C D
A B B B C D D
A A E E C G D
F A F E G G D
F F F E E G G

Waktu pencarian: 504 ms

Banyak kasus yang ditinjau: 141336

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

GG	

9. Contoh 5 × 7 Custom Solvable (Spesifikasi)

```
5 7 5
CUSTOM
...X...
.XXXXX.
XXXXXXX
.XXXXX.
...X...
A
AAA
BB
BBB
CCCC
 C
D
EEE
E
```

```
Masukkan alamat file: Problem_9.txt

      A
  A A A D C
B B B C C C C
  B B E E E
      E

Waktu pencarian: 18 ms

Banyak kasus yang ditinjau: 226

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

10. Contoh 5 × 9 Custom Solvable

```
5 9 7
CUSTOM
....X....
...XXX...
..XXXXX..
.XXXXXXX.
XXXXXXXXX
A
AA
BBB
  B
C
CC
 CC
DD
D
EE
 E
F
FF
G
G
GG
```

```
Masukkan alamat file: Problem_10.txt

        A
      A A B
    C B B B D
  C C E G D D F
C C E E G G G F F

Waktu pencarian: 78978 ms

Banyak kasus yang ditinjau: 26132668

Apakah anda ingin menyimpan solusi? (ya/tidak) ya
```

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	✔	

Repository: github.com/timoruslim/Tucil1_10123053