Mencari Solusi Penyelesaian Permainan IQ Puzzler Pro dengan Algoritma Brute Force

Maheswara Bayu Kaindra - 13523015

kaindramaheswara11@gmail.com, 13523015@std.stei.itb.ac.id

Mengenai Tulisan Secara Umum

IQ Puzzler Pro merupakan suatu permainan puzzle yang bertujuan untuk menguji kemampuan dan kreativitas pemain untuk menyusun balok-balok dengan bentuk sedemikian rupa pada papan permainan dengan aturan sebagai berikut.

1. Seluruh bagian papan harus terisi.
2. Semua balok harus digunakan.
3. Tidak ada bagian balok yang saling bertumpuk dan melewati batas papan permainan.

Tugas kecil pertama Strategi Algoritma ini meminta mahasiswa untuk mencari tepat satu solusi permainan ini menggunakan algoritma brute force. Teknik brute force yang digunakan harus benar-benar murni, tanpa adanya bantuan "pengetahuan" (untuk mengurangi kemungkinan iterasi).

Tulisan ini dibuat sebagai dokumentasi lengkap proses berpikir penulis (Maheswara Bayu Kaindra) dalam mencari solusi persoalan puzzle ini. Secara umum, laporan akan disusun dengan struktur sebagai berikut.

1. Analisis Komponen-komponen Permainan
2. Pendekatan dan Desain Algoritma
3. Algoritma Brute Force untuk Menyelesaikan Permainan
4. Implementasi Program dalam Bahasa Java
5. Demo Program.
6. Lampiran dan Hasil.

Spesifikasi program dan tahapan penyelesaian disusun sejelas mungkin untuk memudahkan pembaca memahami pola pikir penulis (walau tidak sempurna).

Meskipun bahasa pemrograman Java cenderung *object oriented*, penulis tidak melakukan perancangan secara detail, sehingga terdapat beberapa objek yang "disatukan" di dalam satu kelas, misalnya Solver.

Analisis Komponen-komponen Permainan

Papan / Board

Dalam pendekatan ini, papan permainan digambarkan sebagai sebuah array dua dimensi yang terdiri dari karakter-karakter. Ruangan yang terisi akan ditandai dengan karakter huruf kapital (A–Z) yang menandakan bagian balok yang mengisi ruangan tersebut.

Apabila terdapat papan berukuran 3×3, maka papan tersebut digambarkan sebagai berikut.

{'.', '.', '.'},

{'.', '.', '.'},

{'.', '.', '.'}

Fig 01 : Ilustrasi papan permainan untuk papan standard

Ruangan yang belum terisi akan ditandai dengan karakter titik (.). Selain itu, terdapat juga jenis papan custom yang dapat diatur sendiri oleh pengguna, di mana pembatas papan dilambangkan sebagai '*' dan bagian papan yang dapat diisi dilambangkan sebagai '.'.

Balok / Piece

Setiap balok digambarkan sebagai array dua dimensi. Array dua dimensi tersebut disusun sedemikian rupa sehingga berbentuk persegi (atau persegi panjang) dengan ruangan terisi dan ruangan tidak terisi.

Konfigurasi isi ruangan sama dengan konfigurasi pada *papan*, yaitu mengikuti input .txt dan dilengkapi titik (.) sebagai penanda ruangan kosong. Balok-balok tersebut akan disimpan di dalam sebuah *list of pieces*. Berikut merupakan ilustrasi penggambaran balok.

Apabila pada .txt balok berbentuk:

A
AA

Fig 02 : Ilustrasi balok pada .txt

Pada program, balok akan disimpan di dalam array sebagai berikut.

{‘A’, ‘.’},

{‘A’, ‘A’}

Fig 03 : Ilustrasi Penggambaran Balok pada Array 2D

Balok-balok yang sudah tergambar dalam array dua dimensi akan disimpan dalam *list of pieces* dengan masing-masing balok memiliki satu buat list, yang berisi balok itu sendiri beserta segala kemungkinan rotasi dan pencerminannya.

Pendekatan dan Desain Umum Algoritma

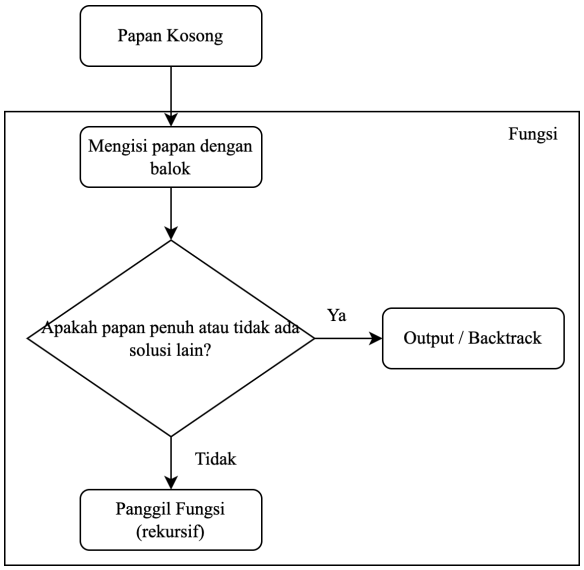
Penggunaan algoritma brute force berarti mencoba semua kemungkinan solusi (baik benar maupun salah) dan mengambil suatu solusi yang dianggap benar. Pada kasus IQ Puzzler Pro, dapat dipastikan bahwa algoritma brute force akan menghasilkan jawaban yang jelas:

1. Terdapat solusi yang memenuhi kedua syarat di atas → menampilkan output solusi persoalan.
2. Tidak terdapat solusi yang memenuhi → menampilkan pesan / *statement*.

Untuk mencapai salah satu dari kedua jawaban tersebut, diperlukan iterasi yang benar-benar menyeluruh untuk semua kemungkinan balok. Oleh karena itu, digunakanlah alur rekursif (paling “sederhana” dan sering digunakan) dengan memanfaatkan *backtracking* untuk mendapatkan semua kemungkinan solusi. Berikut alur kerjanya.

1. Basis ← Program berhenti mencoba kemungkinan apabila
    a. Sudah ditemukan solusi.
    b. Tidak ditemukan solusi setelah mencoba semua kemungkinan.
2. Rekurens ← Program mencari balok lain yang memungkinkan.

Berikut merupakan gambaran desain algoritma menggunakan diagram alur.



Gambar 01 : Diagram Alur Desain Algoritma

Algoritma Brute Force untuk Menyelesaikan Permainan

Berikut merupakan pendekatan brute force untuk menyelesaikan IQ Puzzler Pro dengan rekursif.

1. Setiap block akan dicari seluruh transformasinya dan disimpan dalam sebuah list of Array 2D.
2. Untuk setiap jenis transformasi, dicoba jenis peletakan pertama yang memungkinkan (tanpa peduli apapun, selama blok tersebut tidak menimpa blok lain dan keluar papan, ia akan langsung diletakkan).
3. Setelah balok diletakkan, akan kembali dipanggil prosedur *backtrack* untuk meletakkan balok-balok berikutnya berdasarkan kemungkinan peletakkan pertama yang belum dicoba.
4. Apabila seluruh balok sudah tidak memungkinkan diletakkan pada papan, prosedur akan melakukan return (kembali ke prosedur di luarnya), dan menghapus balok sebelumnya untuk mencoba kemungkinan lain (backtrack).
5. Langkah di atas dilakukan terus menerus sampai ditemukan satu solusi di mana seluruh papan terisi dan semua balok digunakan, atau tidak ditemukan solusi sama sekali.

Implementasi Algoritma dalam Bahasa Java (Spesifikasi Kelas Program)

Terdapat tiga kelas utama yang saling berinteraksi untuk menyelesaikan permainan ini, yaitu *Board*, *Solver*, dan *Main*.

1. Board merupakan kelas yang mengimplementasikan papan permainan sebagai sebuah array 2D yang berisi metode-metode untuk menciptakan papan permainan, menyimpan dan menghapus balok, dan melakukan print papan permainan
    a. `Board(int rows, int cols)` dan `public Board(int rows, int cols)`: Membuat papan permainan berdasarkan ukuran input.
    b. `boolean isFull()`: Memeriksa papan permainan penuh.
    c. `boolean ableToPlaceBlock(char[][] block, int y, int x)`: Memeriksa apakah balok dapat diletakkan pada Papan.
    d. `void placeBlock(char[][] block, int y, int x, char label)`: Meletakkan balok pada papan.
    e. `void removeBlock(char[][] block, int y, int x)`: Menghapus balok dari papan.
    f. Fungsi-fungsi helper.
2. Solver merupakan kelas yang berisi algoritma pembacaan file dan algoritma penyelesaian permainan.
    a. `void solvePuzzle()`: Merupakan prosedur "pembungkus" yang memanggil algoritma pertama kali, mencatat waktu, dan menampilkan jumlah iterasi. Prosedur ini memanggil prosedur *backtrack* (prosedur rekursi sebenarnya).
    b. `void backtrack(int blockIndex)`: Merupakan prosedur utama dalam penyelesaian puzzle yang berisi basis dan rekurens sesuai design algoritma.
    c. `List<char[][]> getAllTransformations(char[][] block)`: Mendapatkan list 8 kemungkinan transformasi, dengan pengecekan setiap loop (agar setiap transformasi berbentuk unik).
    d. Fungsi-fungsi helper.
3. Main, merupakan program utama yang berisi *scanner* untuk meminta input user mengenai nama file input, dan pilihan untuk menyimpan file output.

Board.java

```
public class Board {
   private int rows, cols;
   private char[][] board;
   private static final Map<Character, String> COLOR_MAP = new HashMap<>();
   private static final String RESET = "\u001B[0m";
   private static final String[] COLORS = {
       "\u001B[31m",
```

```
        "\u001B[32m",
        "\u001B[33m",
        "\u001B[34m",
        "\u001B[35m",
        "\u001B[36m",
       "\u001B[91m",
        "\u001B[92m",
        "\u001B[93m",
        "\u001B[94m",
        "\u001B[95m",
        "\u001B[96m"
    };

    public Board(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        board = new char[rows][cols];
        for (char[] row : board) {
            Arrays.fill(row, '.');
        }
    }

    public Board(int rows, int cols, char[][] customBoard){
        this.rows = rows;
        this.cols = cols;
        this.board = customBoard;
    }

    public boolean isFull() {
        for (char[] row : board) {
            for (char cell : row) {
                if (cell == '.') {
                    return false;
                }
            }
        }
        return true;
    }

    public boolean ableToPlaceBlock(char[][] block, int y, int x) {
        int blockRows = block.length;
        int blockCols = block[0].length;

        if ((y + blockRows) > rows || (x + blockCols) > cols) {
            return false;
        }

        for (int i = 0; i < blockRows; ++i) {
            for (int j = 0; j < blockCols; ++j) {
                if (block[i][j] != '.' && board[y + i][x + j] != '.') {
                    return false;
                }
            }
        }

        return true;
    }

    public void placeBlock(char[][] block, int y, int x, char label) {
        for (int i = 0; i < block.length; ++i) {
```

```
            for (int j = 0; j < block[i].length; ++j) {
                if (block[i][j] != '.') {
                    board[y + i][x + j] = label;
                }
            }
        }
    }

    public void removeBlock(char[][] block, int y, int x) {
        for (int i = 0; i < block.length; ++i) {
            for (int j = 0; j < block[i].length; ++j) {
                if (block[i][j] != '.') {
                    board[y + i][x + j] = '.';
                }
            }
        }
    }

    public void printBoard() {
        assignColors();
        for (char[] row : board) {
            for (char cell : row) {
                if (cell == '.') {
                    System.out.print(RESET + "." + RESET);
                } else if (cell == '*'){
                    System.out.print(RESET + ' ' + RESET);
                } else if (cell != '\0') {
                    System.out.print(COLOR_MAP.get(cell) + cell + RESET);
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
        System.out.println();
    }

    private void assignColors() {
        if (!COLOR_MAP.isEmpty()) return;
        int colorIndex = 0;
        for (int i = 0; i < Solver.boardBlocks.size(); i++) {
            char blockLabel = (char) ('A' + i);
            COLOR_MAP.put(blockLabel, COLORS[colorIndex % COLORS.length]);
            colorIndex++;
        }
    }

    public void saveResult(String filePath){
        try (PrintWriter writer = new PrintWriter(new FileWriter(filePath))) {
            for (int i = 0; i < board.length; i++) {
                for (int j = 0; j < board[i].length; j++) {
                    writer.print(board[i][j]);
                }
                writer.println();
            }
        } catch (IOException e) {

        }
    }
}
```

Solver.java

```
public class Solver {
   public static int N, M, P;
   public static String boardType;
   public static List<char[][]> boardBlocks = new ArrayList<>();
   public static long executionTime;
   public static int steps;
   public static Board board;
   private static boolean solutionFound = false;

   public static void solvePuzzle(){
       long initialTime = System.nanoTime();
       backtrack(0);
       long endTime = System.nanoTime();

       executionTime = (endTime - initialTime)/1000000;
       if (solutionFound) {
           System.out.println("Solusi ditemukan.\n");
           board.printBoard();
           System.out.println("Waktu eksekusi: " + Solver.executionTime + " ms.\n");
           System.out.println("Percobaan: " + Solver.steps + ".");
       } else {
           System.out.println("Tidak ada solusi yang ditemukan.\n");
           System.out.println("Waktu eksekusi: " + Solver.executionTime + " ms.\n");
           System.out.println("Percobaan: " + steps + ".");
       }
   }

   private static void backtrack(int blockIndex){
       if (blockIndex == boardBlocks.size()){
           if (board.isFull()){
               solutionFound = true;
           }
           return;
       } else {
           char[][] block = boardBlocks.get(blockIndex);
           char blockName = (char) ('A' + blockIndex);

           for (int y = 0; y < N; ++y){
               for (int x = 0; x < M; ++x){
                   List<char[][]> allTransformations = getAllTransformations(block);
                   for (char[][] transformedBlock : allTransformations){
                       steps++;
                       if (board.ableToPlaceBlock(transformedBlock, y, x)){
                           board.placeBlock(transformedBlock, y, x, blockName);

                           backtrack(blockIndex + 1);

                           if (solutionFound) return;

                           board.removeBlock(transformedBlock, y, x);
                       }
                   }
               }
           }
       }
   }

   private static List<char[][]> getAllTransformations(char[][] block){
       List<char[][]> transformedBlocks = new ArrayList<>();
       transformedBlocks.add(block);
```

```
        char[][] rotatedBlock = block;
        for (int i = 0; i < 3; ++i){
            rotatedBlock = rotate(rotatedBlock);
            if (!isFoundInList(transformedBlocks, rotatedBlock)){
                transformedBlocks.add(rotatedBlock);
            }
        }

        char[][] mirroredBlock = mirror(block);
        if (!isFoundInList(transformedBlocks, mirroredBlock)){
            transformedBlocks.add(mirroredBlock);
        }
        for (int i = 0; i < 3; ++i){
            mirroredBlock = rotate(mirroredBlock);
            if (!isFoundInList(transformedBlocks, mirroredBlock)){
                transformedBlocks.add(mirroredBlock);
            }
        }

        return transformedBlocks;
    }

    private static boolean isFoundInList(List<char[][]> list, char[][] block){
        for (char[][] blockInList : list){
            if (isEqual(blockInList, block)){
                return true;
            }
        }
        return false;
    }

    private static boolean isEqual(char[][] block1, char[][] block2){
                    if   ((block1.length   !=   block2.length)   ||   (block1[0].length   !=
block2[0].length)){
            return false;
        }

        for (int i = 0; i < block1.length; i++){
            if (!Arrays.equals(block1[i], block2[i])){
                return false;
            }
        }

        return true;
    }

    private static char[][] rotate(char[][] block){
        int rows = block.length;
        int cols = block[0].length;
        char[][] rotated = new char[cols][rows];

        for (int i = 0; i < rows; ++i){
            for (int j = 0; j < cols; ++j){
                rotated[j][rows - 1 - i] = block[i][j];
            }
        }

        return rotated;
    }
```

```
    private static char[][] mirror(char[][] block){
        int rows = block.length;
        int cols = block[0].length;
        char[][] mirrored = new char[rows][cols];

        for (int i = 0; i < rows; ++i){
            for (int j = 0; j < cols; ++j){
                mirrored[i][cols - 1 - j] = block[i][j];
            }
        }

        return mirrored;
    }

    public static void readInput(String filename) throws IOException {
        BufferedReader br = new BufferedReader(new java.io.FileReader(filename));

        String[] boardSpesification = br.readLine().split(" ");
        N = Integer.parseInt(boardSpesification[0]);
        M = Integer.parseInt(boardSpesification[1]);
        P = Integer.parseInt(boardSpesification[2]);

        boardType = br.readLine();

        char[][] customBoard = new char[N][M];
        if (boardType.equals("CUSTOM")){
            for (int i = 0; i < N; ++i){
                String line = br.readLine().trim();
                for (int j = 0; j < M; ++j){
                    customBoard[i][j] = line.charAt(j);
                }
            }
        }

        List<String> lines = new ArrayList<>();
        String line;
        char tempBlock = '\0';
        while ((line = br.readLine()) != null){
            if (!line.isEmpty()){
                char currentBlock = getCurrentBlock(line);
                if (tempBlock != '\0' && currentBlock != tempBlock){
                    boardBlocks.add(convertToBlock(lines));
                    lines.clear();
                }
                lines.add(line);
                tempBlock = currentBlock;
            }
        }
        boardBlocks.add(convertToBlock(lines));

        if(boardType.equals("DEFAULT")){
            board = new Board(N, M);
        } else {
            board = new Board(N, M, customBoard);
        }

        br.close();
    }

    private static char getCurrentBlock(String line){
        char currentBlock = ' ';
        int rows = line.length();
```

```
        int i = 0;

        while (i < rows && currentBlock == ' '){
            currentBlock = line.charAt(i);
            i++;
        }
        return currentBlock;
    }

    private static char[][] convertToBlock(List<String> lines) {
        int rows = lines.size();
        int cols = lines.stream().mapToInt(String::length).max().orElse(0);
        char[][] block = new char[rows][cols];

        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                if (j < lines.get(i).length() && lines.get(i).charAt(j) != ' ') {
                    block[i][j] = lines.get(i).charAt(j);
                } else {
                    block[i][j] = '.';
                }
            }
        }

        return block;
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Masukkan nama file test case: ");
        String filename = scanner.nextLine().trim();

        String filePath = "../test/" + filename;
        String outputPath = "../test/solution-" + filename;

        try {
            Solver.readInput(filePath);
            Solver.solvePuzzle();

            System.out.print("Simpan solusi / state terakhir board? (ya/tidak): ");
            String response = scanner.nextLine().trim();

            if (response.equals("ya")){
                Solver.board.saveResult(outputPath);
                System.out.println("Solusi disimpan pada: " + outputPath);
            }
        } catch (IOException e) {
            System.out.println("Terjadi kesalahan saat membaca file.");
        }
        scanner.close();
    }
}
```

Demo Program dan Pengujian Algoritma

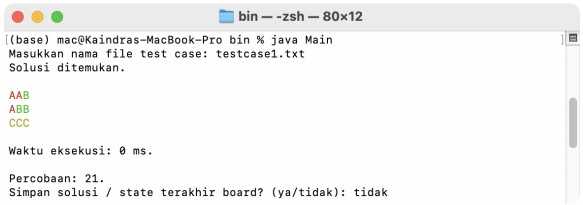
Kasus 1 : Kasus 3x3 Sederhana (terdapat solusi)

testcase1.txt

```
3 3 3
DEFAULT
AA
A
BB
B
CCC
```

Fig 04 : Test Case 1

Hasil eksekusi:



Gambar 02 : Hasil Test Case 1

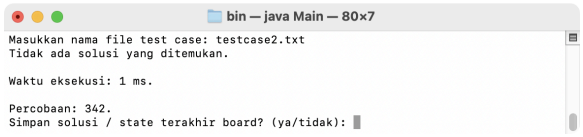
Kasus 2 : Kasus 3x3 Sederhana (tidak terdapat hasil karena ada papan kosong)

testcase2.txt

```
3 3 3
DEFAULT
AAA
BBB
CC
```

Fig 05 : Test Case 2

Hasil eksekusi:



Gambar 03 : Hasil Test Case 2

Kasus 3 : Kasus 5x5 (terdapat hasil)

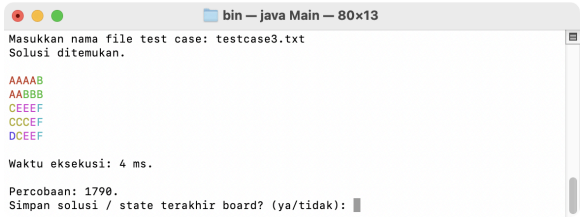
testcase3.txt

```
5 5 7
DEFAULT
AAAA
AA
BBB
B
C
CCC

 C
D
EEE
  E
 EE
F
F
F
```

Fig 06 : Test Case 3

Hasil eksekusi:



Gambar 04 : Hasil Test Case 3

Kasus 4: 5x5 (tidak terdapat hasil luas blok melebihi luas papan)

testcase4.txt

```
5 5 7
DEFAULT
AAAA
AA
BBB
B
C
CCC
 C
D
EEE
  E
 EE
F
F
F
F
```

Fig 07 : Test Case 4

Hasil eksekusi:

Gambar 05 : Hasil Test Case 4

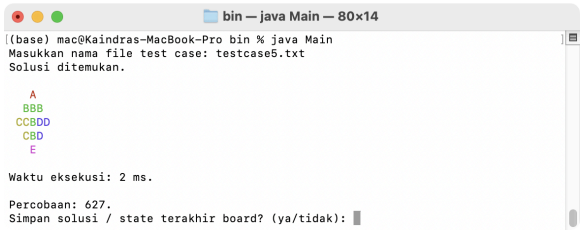
Kasus 5 : Custom Board (terdapat hasil)

testcase5.txt

```
5 7 5
CUSTOM
***.***
**...**
*.....*
**...**
***.***
A
BBB
 B
 B
CC
 C
DD
D
E
```

Fig 08 : Test Case 5

Hasil eksekusi:



Gambar 06 : Hasil Test Case 5

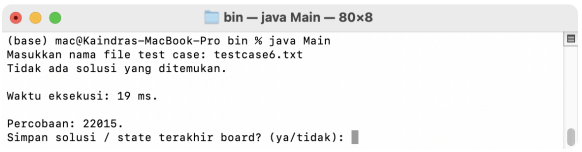
Kasus 6 : Custom Board (tidak terdapat hasil karena terdapat bagian papan kosong)

testcase6.txt

```
5 7 4
CUSTOM
***.***
**...**
*.....*
**...**
***.***
A
BBB
 B
 B
CC
 C
DD
D
```

Fix 09 : Test case 6

Hasil eksekusi:



Gambar 07 : Hasil Test Case 6

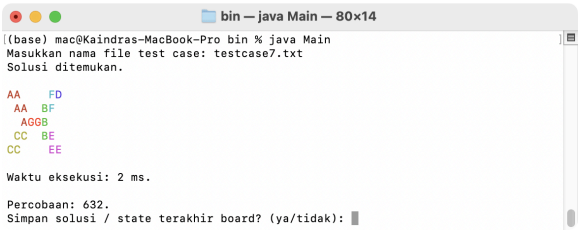
Kasus 7 : Custom Board Silang (terdapat hasil)

testcase7.txt

```
5 8 7
CUSTOM
..****..
*..**..*
**....**
*..**..*
..****..
AA
 AA
  A
BBB
CC
 CC
D
EE
 E
FF
GG
```

Fig 10 : Test Case 7

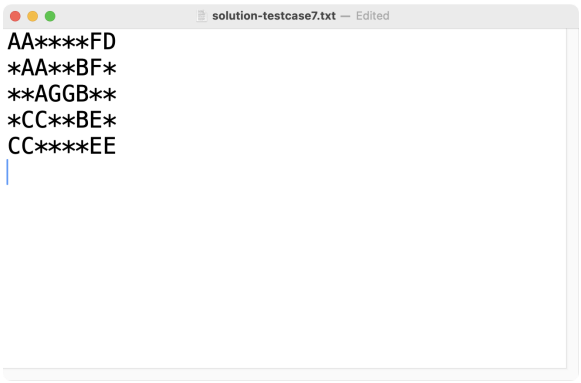
Hasil eksekusi:



Gambar 08 : Hasil Test Case 7

Setelah program dijalankan, user dapat menyimpan solusi dari setiap kasus dengan mengetikkan “ya” ketika program meminta input.

Apabila user memilih ‘ya’, maka file solusi akan tersimpan pada direktori *test* dengan nama *solusi-<nama-file>.txt*. Seperti contohnya apabila user menyimpan solusi test case 7, maka berikut merupakan tampilan solution-*testcase7.txt*.



Gambar 09 : Output Solusi Test Case 7 Jika Disimpan

Apabila test case tidak memiliki solusi, maka output file solusi akan berupa papan kosong.

Gambar 10 : Output jika Program Gagal Menemukan Solusi

Lampiran (Repositori)

Seluruh kode program disimpan di dalam repositori Github dengan keterangan sebagai berikut

- Nama: Tucil1_13523015
- Struktur Folder:
    - doc: dokumentasi dan paper
    - src: source code
    - bin: executable program
    - test: test case dan output file
    - README.md
- Tautan: Github (https://github.com/MaheswaraKaindra/Tucil1_13523015.git)

Tabel Keberhasilan Program

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	☑	
2	Program berhasil dijalankan	☑	
3	Solusi yang diberikan program benar dan memenuhi 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		☑
9	Program dibuat oleh saya sendiri	☑	