LAPORAN TUGAS KECIL I
IF2211 - STRATEGI ALGORITMA

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Disusun oleh :

Andrew Isra Saputra DB

13523110

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

DAFTAR ISI ii
BAB 1 1
DESKRIPSI MASALAH 1
BAB 2 3
TEORI SINGKAT 3
2.1 Algoritma Brute Force 3
2.2 Backtracking 5
Prinsip Dasar Backtracking: 5
Contoh Penggunaan Backtracking: 5
BAB 3 6
ALGORITMA BRUTE FORCE DALAM PENYELESAIAN PERMAINAN 6
BAB 4 7
SOURCE CODE 7
BAB 5 8
TESTING 8
BAB 6 10
LAMPIRAN 10
6.1 Tautan Github 10
6.2 Tabel Pengecekan 10

BAB 1
DESKRIPSI MASALAH

IQ Puzzle Pro merupakan salah satu permainan untuk mengasah kemampuan kognitif dengan menguji konsentrasi, pola pikir, serta strategi untuk pemecahannya. Permainan ini dapat dimainkan oleh rentang umur berapa saja, baik anak-anak maupun orang dewasa sekalipun. IQ Puzzle Pro memiliki level kesulitan yang beragam, dari yang paling mudah dengan dimensi 2D dan ukuran yang kecil hingga paling sulit dengan dimensi 3D dan ukuran yang cukup besar. Hal ini yang membuat IQ Puzzle Pro dijadikan salah satu permainan yang sangat relevan untuk menilai perkembangan kognitif manusia.

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh perusahaan Smart Games. Tujuan dari permainan ini adalah pemain harus dapat mengisi seluruh papan dengan piece (blok puzzle) yang telah tersedia.

Komponen penting dari permainan IQ Puzzler Pro terdiri dari:

1. **Board (Papan)** – Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.
2. **Blok/Piece** – Blok adalah komponen yang digunakan pemain untuk mengisi papan kosong hingga terisi penuh. Setiap blok memiliki bentuk yang unik dan semua blok harus

digunakan untuk menyelesaikan puzzle.

Permainan dimulai dengan papan yang kosong. Pemain dapat meletakkan blok puzzle sedemikian sehingga tidak ada blok yang bertumpang tindih (kecuali dalam kasus 3D). Setiap blok puzzle dapat dirotasikan maupun dicerminkan. Puzzle dinyatakan selesai jika dan hanya jika papan terisi penuh dan seluruh blok puzzle berhasil diletakkan.

BAB 2
TEORI SINGKAT

2.1 Algoritma *Brute Force*

Algoritma *Brute force* adalah pendekatan yang lempeng (*straightforward*) untuk memecahkan suatu persoalan. Algoritma ini didasarkan pada pernyataan pada persoalan (problem statement) dan Definisi/konsep yang dilibatkan. Algoritma ini memiliki metode pencarian solusi dengan mencoba semua kemungkinan secara sistematis hingga menemukan hasil yang benar. Teknik ini bekerja dengan melakukan eksplorasi menyeluruh terhadap ruang solusi tanpa mempertimbangkan optimasi atau heuristik tertentu.

Contoh dari algoritma Brute force adalah sebagai berikut :

1. Mencari elemen terbesar (terkecil)
2. Mencari nilai rata-rata pada array
3. Pencarian beruntun (Sequential Search)

4. Menghitung nilai eksponen, faktorial, perkalian matriks, uji bilangan prima, logika pengurutan (sorting), seperti bubble sort dan selection sort, pencocokan string (String Matching/Pattern Matching)

Karakteristik Algoritma Brute Force

1. Algoritma Brute Force umumnya tidak “cerdas” dan tidak efisien, karena dalam penyelesaiannya membutuhkan jumlah langkah yang besar. Kata “force” sendiri lebih mengedepankan “tenaga” ketimbang “otak”. Kadang-kadang Algoritma Brute Force disebut juga Algoritma Naif (naïve algorithm).
2. Algoritma Brute Force lebih cocok untuk masalah yang berukuran kecil. Karna sederhana, dan implementasinya mudah.
3. Algoritma Brute Force sering digunakan sebagai basis pembanding dengan Algoritma yang lebih efisien.

4. Meskipun bukan metode yang efisien, hampir semua masalah dapat diselesaikan dengan Algoritma Brute Force. Bahkan, ada masalah yang hanya dapat diselesaikan dengan metode Brute Force. Contoh : mencari elemen terbesar di dalam senarai. Atau menghitung jumlah dari n buah bilangan.
5. Karna ketidakefisiennya, Algoritma Brute Force dapat mencari pola-pola yang mendasar, keteraturan , atau trik-trik khusus, yang biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih efisien.
6. Untuk kecil, kesederhanaan Algoritma Brute Force biasanya lebih diperhitungkan dari pada ketidakefisiennya karena Algoritma Brute Force sering digunakan sebagai basis, bila membandingkan beberapa alternatif algoritma yang efisien.
7. Selain itu, Algoritma Brute Force seringkali lebih mudah diimplementasikan dari pada Algoritma yang lebih canggih karena kesederhanaannya, kadang-kadang juga Algoritma Brute Force dapat lebih efisien (ditinjau dari segi implementasi).

Kelebihan Algoritma Brute Force

1. Algoritma Brute Force dapat digunakan untuk memecahkan hampir sebagian besar masalah (wide applicability).
2. Algoritma Brute Force mudah dimengerti.
3. Algoritma Brute Force menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan string, perkalian matriks.
4. Algoritma Brute Force menghasilkan algoritma baku (standard) untuk tugas-tugas komputasi, seperti penjumlahan/perkalian n buah bilangan, menentukan elemen minimum atau maksimum di dalam tabel (list).

Kekurangan Algoritma Brute Force

1. Algoritma Brute Force jarang menghasilkan algoritma yang efisien.
2. Beberapa Algoritma Brute Force lambat sehingga tidak dapat diterima.
3. Algoritma Brute Force Tidak sekontruktif/sekreatif teknik pemecahan masalah lainnya.

(sumber: ;

https://faris6593.blogspot.com/2013/10/karakteristik-kelebihan-kelemahan-algoritma-brute-force.html - Berbagi Pengetahuan)

2.2 *Backtracking*

Backtracking adalah teknik algoritmik untuk mencari solusi dengan cara mencoba satu per satu kemungkinan dan membatalkan (*backtrack*) jika suatu pilihan tidak menghasilkan solusi yang valid. Teknik ini sering digunakan untuk menyelesaikan masalah kombinatorial dan eksplorasi ruang pencarian secara sistematis.

Prinsip Dasar Backtracking:

1. Memulai dari satu titik awal.
2. Mencoba satu pilihan dan melanjutkan ke langkah berikutnya.
3. Jika menemukan solusi yang valid, lanjutkan. Jika tidak, kembali ke langkah sebelumnya (*backtrack*) dan coba pilihan lain.
4. Proses ini terus berlanjut sampai semua kemungkinan dicoba atau solusi ditemukan.

Contoh Penggunaan Backtracking:

1. Pencarian jalur dalam labirin
2. Permainan Sudoku
3. N-Queens Problem
4. Pencarian subset atau permutasi dari suatu himpunan
5. Permasalahan IQ Puzzle Pro

BAB 3

ALGORITMA BRUTE FORCE DALAM PENYELESAIAN PERMAINAN

Berikut merupakan kode untuk algoritma *brute force* yang digunakan. Sesuai dengan prinsipnya, ide dari pemecahan masalah tersebut dengan mencoba semua kemungkinan yang ada, sambil menjalankan waktu eksekusi dan menghitung berapa iterasi yang digunakan.

Seperti yang diketahui pada spesifikasi, blok dapat ditransformasi dengan rotasi 0°, 90°, 180°, dan 270° serta pencerminan dengan sumbu vertikal (*flip horizontal)*. Pada kode, hanya diberikan method *flip horizontal* (bisa dilihat pada source code lengkap di github https://github.com/andrewisra/Tucil1_13523110) karena transformasi *flip vertikal* dibangun dari kombinasi *flip horizontal* dan rotasi 90°. Sehingga terdapat 2*4 kemungkinan maksimal variasi untuk satu blok.

Program akan melakukan *looping* dengan memanggil fungsi solve(0) yang merupakan inisialisasi awal untuk rekursi pengecekan kasus (menggunakan *backtracking*) dan berhenti (base case) ketika index telah mencapai nilai p (variabel penyimpan banyaknya blok). Jika berhasil, program akan menyimpan boardGame tersebut dan mengembalikan True.

Terdapat sedikit kekurangan pada kode ini, yaitu:

1. Defenisi “banyak kasus yang ditinjau” berlandaskan seberapa banyak program melakukan method placeBlock dalam artian seberapa banyak program mencoba memasukkan blok. Sehingga hasil dari perhitungan akan lebih banyak dibanding kasus aslinya

Pembacaan file input harus diberi *enter space* untuk memisahkan setiap blok

BAB 4

SOURCE CODE

```java
import java.util.List;
import java.io.*;

public class Solver {
    private GameBoard board;
    private List<Block> blocks;
    private long startTime;
    private int iterationCount = 0;

    public Solver(GameBoard board, List<Block> blocks) {
        this.board = board;
        this.blocks = blocks;
    }

    public boolean solve(int index) {
        if (index == blocks.size()) {
            if (board.isFull()) {
                System.out.println("Solusi ditemukan:");
                board.printBoardWithColor();
                return true;
            }
            return false;
        }

        Block block = blocks.get(index);

        for (int x = 0; x < board.getN(); x++) {
            for (int y = 0; y < board.getM(); y++) {
                for (int r = 0; r < 4; r++) { // Coba 4 rotasi
                    if (board.canPlace(block, x, y)) {
                        board.placeBlock(block, x, y);
                        iterationCount++;

                        if (solve(index + 1)) {
                            return true;
                        }
                        board.removeBlock(block, x, y); // Backtracking
                        block.rotate90();
                    }
                }
                block.flipHorizontal(); // Coba dengan cerminan

                for (int r = 0; r < 4; r++) { // Coba 4 rotasi lagi
                    if (board.canPlace(block, x, y)) {
                        board.placeBlock(block, x, y);
                        iterationCount++;

                        if (solve(index + 1)) {
                            return true;
                        }
                        board.removeBlock(block, x, y); // Backtracking
                        block.rotate90();
                    }
                }

            }
        }
        return false;
    }

    public void solvePuzzle() {
        startTime = System.currentTimeMillis();
        if (!solve(0)) {
            System.out.println("Tidak ada solusi ditemukan.");
        }
        long executionTime = System.currentTimeMillis() - startTime;
        System.out.println("Waktu pencarian: " + executionTime + " ms");
        System.out.println();
        System.out.println("Banyak kasus yang ditinjau: " + iterationCount);
        System.out.println();
    }
```

```java
    public void saveSolution(String filename) {
        try {
            // Buat folder test jika belum ada
            File directory = new File("test");
            if (!directory.exists()) {
                directory.mkdir(); // Buat folder test
            }

                // Simpan file di dalam folder test/
            File outputFile = new File("test/" + filename);
            BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile));

                // Simpan konfigurasi solusi (papan permainan)
            for (char[] row : board.getBoard()) {
                writer.write(new String(row));
                writer.newLine();
            }

            // Simpan metadata waktu eksekusi dan iterasi
            writer.write("Waktu pencarian: " + executionTime + " ms");
            writer.newLine();
            writer.write("Banyak kasus yang ditinjau: " + iterationCount);
            writer.newLine();

            writer.close();
            System.out.println("Solusi berhasil disimpan di: test/" + filename);
        } catch (IOException e) {
            System.out.println("Gagal menyimpan solusi: " + e.getMessage());
        }
    }


    private long executionTime;

}

```

BAB 5

TESTING

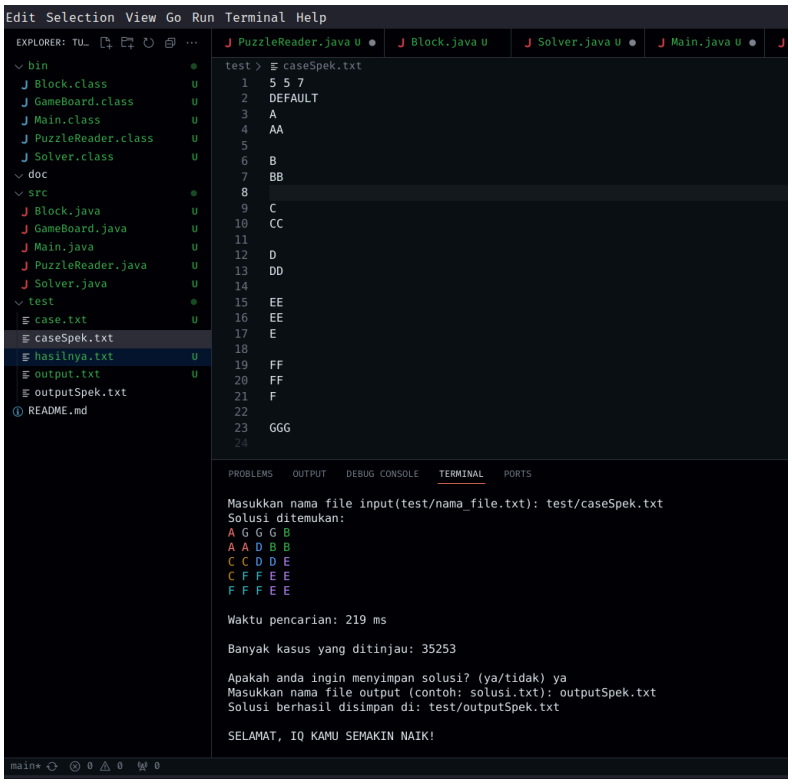
1. Test Case 1

test > case.txt

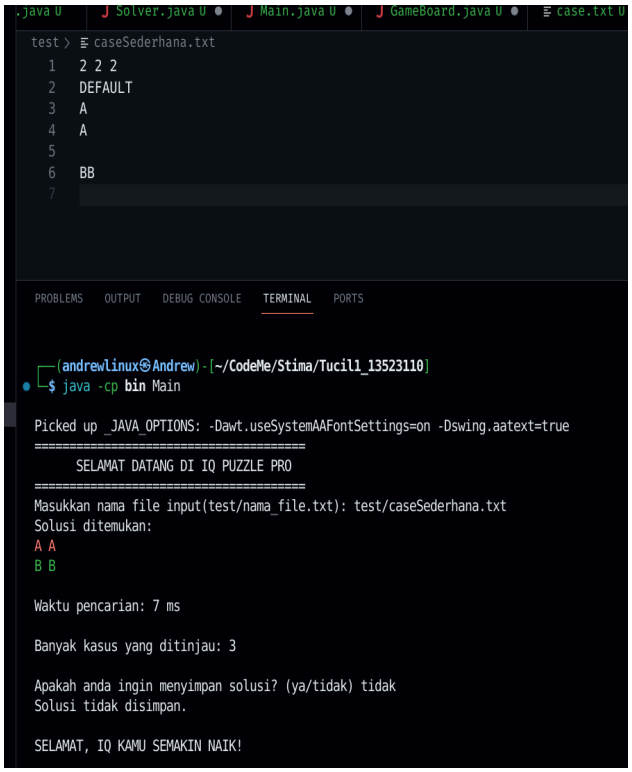
```
5 10 8
DEFAULT
A
A
A
A
A

BBBBB
BBBBB
BBBBB

CCCCC

DDDDD

E
E
E
E
E

F

GGGG

IIIII
IIIII
```

```
┌──(andrewlinux㉿Andrew)-[~/CodeMe/Stima/Tucil1_13523110]
└─$ java -cp bin Main

Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
=====================================
      SELAMAT DATANG DI IQ PUZZLE PRO
=====================================
Masukkan nama file input(test/nama_file.txt): test/case.txt
Solusi ditemukan:
A B B B C D D D D D
A B B B C E E E E E
A B B B C F G G G G
A B B B C I I I I I
A B B B C I I I I I

Waktu pencarian: 41 ms

Banyak kasus yang ditinjau: 133
```

2. Test Case 2



3. Test Case 3



BAB 6

LAMPIRAN

6.1 Tautan Github

https://github.com/andrewisra/Tucil1_13523110

6.2 Tabel Pengecekan

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	✓	