

TUGAS KECIL 1

IF-2230

STRATEGI ALGORITMA



Haegen Quinston

13523109

Dosen Pengampu:

Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

Dr. Ir. Rinaldi Munir, M.T.

Monterico Adrian, S.T, M.T.

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025**

DAFTAR ISI

DAFTAR ISI	2
BAB 1 ALGORITMA	3
1.1 Algoritma Brute Force	3
1.2 Aplikasi pada Program	3
1.3 Pseudocode Algoritma	4
BAB 2 PROGRAM	5
2.1 Modul & Library	5
2.2 Source Code	6
BAB 3 EKSPERIMEN	11
LAMPIRAN	15
Link Repository	15
Checklist	15

BAB 1 ALGORITMA

1.1 Algoritma Brute Force

Algoritma Brute Force adalah suatu algoritma skema pencarian yang sederhana dan komprehensif yang secara sistematis meninjau seluruh kemungkinan dari permasalahan hingga jawaban ditemukan. Pendekatan ini umumnya digunakan untuk pemecahan masalah berskala kecil sehingga memungkinkan penyelidikan mendalam. Namun, karena kompleksitas waktu tinggi, pendekatan ini tidak cocok untuk memecahkan masalah berskala besar.

1.2 Aplikasi pada Program

Dalam penyelesaian permainan Block Puzzle, program yang dibuat adalah sebagai berikut:

Input Awal

Pengguna memasukkan data berupa dimensi papan (tinggi h dan lebar w) serta jumlah block n . Papan tersebut direpresentasikan sebagai struktur data dua dimensi, sedangkan setiap block merupakan objek dengan bentuk tertentu.

Pencarian Solusi dengan Backtracking

- **Iterasi Seluruh Papan**
Program akan melakukan iterasi untuk setiap sel pada papan. Total iterasi yang dilakukan adalah sekitar $h*w*n$, karena pada tiap posisi papan, program mencoba meletakkan setiap block yang tersedia.
- **Variasi Block**
Untuk setiap block yang belum terpakai, ada 8 variasi konfigurasi yang harus dicoba: 4 kemungkinan rotasi (0° , 90° , 180° , 270°) dan 2 kondisi (versi asli dan versi mirror). Dengan demikian, setiap block memiliki 8 konfigurasi potensial.
- **Rekursi**
Jika block valid untuk ditempatkan, maka block tersebut diletakkan pada papan dan statusnya ditandai sebagai “terpakai”. Program kemudian memanggil dirinya sendiri (rekursif) untuk mencoba menempatkan block selanjutnya.
- **Basis Rekursi**
Fungsi rekursif memiliki basis: jika semua block telah ditempatkan dan papan sepenuhnya terisi, maka solusi telah ditemukan, dan fungsi mengembalikan nilai True. Jika tidak, atau tidak ada konfigurasi yang berhasil, maka fungsi mengembalikan False sehingga terjadi backtracking (block yang salah diletakkan dihapus dan statusnya direset).

Setelah algoritma menemukan solusi pertama untuk permasalahan Block Puzzle, proses komputasi akan berhenti dan program langsung berpindah ke menu output. Pada menu ini, ditampilkan informasi berupa jumlah kasus yang telah diperiksa, waktu eksekusi yang dibutuhkan, tampilan solusi, serta opsi untuk mengunduh hasil.

Secara algoritmis, perkiraan jumlah iterasi utama yang dilakukan adalah sebanyak $(h*w*8)^n * n!$ kali. di mana h dan w merupakan dimensi papan dan n adalah jumlah blok. Perhitungan tersebut belum menghitung iterasi-iterasi tambahan seperti pengecekan kesesuaian blok dengan papan, pemasangan blok, proses backtracking (pengambilan kembali blok), serta cloning blok.

Karena algoritma yang digunakan mencoba semua kemungkinan solusi secara menyeluruh, maka untuk permasalahan dengan skala besar, pendekatan ini kurang efisien—sesuai dengan teori kompleksitas komputasi.

1.3 Pseudocode Algoritma

Pseudocode di bawah ini dibuat untuk merepresentasikan algoritma brute force yang dibuat dalam notasi algoritmik. Sebagai catatan, pseudocode ini masih mengandung banyak metode di luar algoritma utama (terdapat pada objek lain) yang didefinisikan pada kode namun tidak dibuat rincian algoritma/metodenya.

```
extern integer iterations

function findEmptyCell(input Board board) -> output integer[]
function allBlocksUsed(input boolean[]used) -> output boolean
function canPlace(input Block block, input Board board, input
integer row, input integer col) -> output boolean
procedure placeBlock(input Block block, input/output Board board,
input integer row, input integer col)
procedure removeBlock(input Block block, input/output Board board,
input integer row, input integer col)

function solve(input Board board, input Block[] blocks, input
boolean[] used) -> output boolean
# KAMUS
row, col, i, j: integer
orientedBlock: Block

# ALGORITMA
row traversal (0...board.getHeight())
  col traversal (0...board.getWidth())
    if allBlocksUsed(used) then
      if findEmptyCell(board) = null then
        -> true
      else
        -> false
    i traversal (0...blocks.length)
      if not used[i] then
        j traversal (0...8)
          orientedBlock <- blocks[i].orientations[j]
          if canPlace(orientedBlock, board, row, col) then
            placeBlock(orientedBlock, board, row, col)
            used[i] <- true
```

```
        if solve(board, blocks, used) then
            -> true
        removeBlock(orientedBlock, board, row, col)
        used[i] <- false
iterations <- iterations + 1
-> false
```

BAB 2 PROGRAM

2.1 Modul & Library

Program ini ditulis dalam bahasa Java, menggunakan library eksternal sebagai berikut:

1. Scanner
2. File
3. FileReader
4. ArrayList
5. Hashmap
6. Map
7. IOException
8. FileNotFoundException
9. InputMismatchException
10. NoSuchElementException

Program ini menggunakan view/paradigma object-oriented, dimana modul-modul dibagi berdasarkan sifat dan fungsionalitasnya. Pembagian kelas dan file adalah sebagai berikut:

- Main.java (main program)
- Block.java
- Board.java
- Bruteforce.java (brute force solving method)
- InputHandler.java

2.2 Source Code

Source code lengkap dari program tidak ditampilkan secara utuh karena telah tersedia melalui tautan repository Github yang terlampir. Yang ditampilkan hanya bagian terpenting, yaitu kode algoritma inti program (bruteforce) dan kode objek Block (memiliki iterasi rotasi).

Source Code Bruteforce.java

```
public class Bruteforce {
    public static long iterations = 1;
    public static boolean solve(Board board, Block[] blocks, boolean[]
used) {
        for (int row = 0; row < board.getHeight(); row++) {
            for (int col = 0; col < board.getWidth(); col++) {
                if (allBlocksUsed(used)) {
                    if (findEmptyCell(board) == null) {
                        return true;
                    }
                    return false;
                }
                for (int i = 0; i < blocks.length; i++) {
                    if (!used[i]) {
                        for (Block orientedBlock :
blocks[i].orientations) {
                            if (canPlace(orientedBlock, board, row, col))
{
                                placeBlock(orientedBlock, board, row,
col);

                                used[i] = true;
                                if (solve(board, blocks, used)) {
                                    return true;
                                }
                                removeBlock(orientedBlock, board, row,
col);

                                used[i] = false;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```

        }

    }

}

iterations++;
return false;
}

private static int[] findEmptyCell(Board board) {
    for (int i = 0; i < board.getHeight(); i++) {
        for (int j = 0; j < board.getWidth(); j++) {
            if (board.getElmt(i, j) == 0) {
                return new int[]{i, j};
            }
        }
    }
    return null;
}

private static boolean canPlace(Block block, Board board, int row,
int col) {
    if (row + block.getHeight() > board.getHeight() || col +
block.getWidth() > board.getWidth()) {
        return false;
    }
    for (int i = 0; i < block.getHeight(); i++) {
        for (int j = 0; j < block.getWidth(); j++) {
            if (block.getElmt(i, j) == 1 && board.getElmt(row + i,
col + j) == 1) {
                return false;
            }
        }
    }
    return true;
}

private static void placeBlock(Block block, Board board, int row, int
col) {
    for (int i = 0; i < block.getHeight(); i++) {
        for (int j = 0; j < block.getWidth(); j++) {

```

```

        if (block.getElmt(i, j) == 1) {
            board.setChar(row + i, col + j, block.getType());
        }
    }
}

private static void removeBlock(Block block, Board board, int row,
int col) {
    for (int i = 0; i < block.getHeight(); i++) {
        for (int j = 0; j < block.getWidth(); j++) {
            if (block.getElmt(i, j) == 1) {
                board.removeChar(row + i, col + j);
            }
        }
    }
}

private static boolean allBlocksUsed(boolean[] used) {
    for (boolean b : used) {
        if (!b) {
            return false;
        }
    }
    return true;
}
}

```

Source Code Block.java

```

import java.util.ArrayList;

public class Block {
    private int height;
    private int width;
    public int[][] shape;
    private char type;
    public ArrayList<Block> orientations = new ArrayList<>();

    public void create(int h, int w) {

```



```
        this.height = h;
        this.width = w;
        shape = new int[height][width];
    }

    public void set(int h, int w, int value) {
        shape[h][w] = value;
    }

    public void setType(char t) {
        type = t;
    }

    public char getType() {
        return type;
    }

    public int getHeight() {
        return height;
    }

    public int getWidth() {
        return width;
    }

    public int getElmt(int h, int w) {
        return shape[h][w];
    }

    public Block cloneBlock() {
        Block newBlock = new Block();
        newBlock.create(this.height, this.width);
        for (int i = 0; i < this.height; i++) {
            for (int j = 0; j < this.width; j++) {
                newBlock.set(i, j, this.shape[i][j]);
            }
        }
        newBlock.setType(this.type);
        return newBlock;
    }
}
```

```

    }

    public Block getRotated() {
        Block newBlock = new Block();
        newBlock.create(this.width, this.height);
        for (int i = 0; i < this.height; i++) {
            for (int j = 0; j < this.width; j++) {
                newBlock.set(j, this.height - i - 1, this.shape[i][j]);
            }
        }
        newBlock.setType(this.type);
        return newBlock;
    }

    public Block getMirrored() {
        Block newBlock = new Block();
        newBlock.create(this.height, this.width);
        for (int i = 0; i < this.height; i++) {
            for (int j = 0; j < this.width; j++) {
                newBlock.set(i, this.width - j - 1, this.shape[i][j]);
            }
        }
        newBlock.setType(this.type);
        return newBlock;
    }

    public void computeOrientations() {
        orientations.clear();
        Block original = this.cloneBlock();
        orientations.add(original);
        Block current = original;
        for (int i = 1; i < 4; i++) {
            current = current.getRotated();
            orientations.add(current);
        }
        Block mirrored = this.getMirrored();
        orientations.add(mirrored);
        current = mirrored;
        for (int i = 1; i < 4; i++) {

```

```

        current = current.getRotated();
        orientations.add(current);
    }
}

public void print() {
    for (int i = 0; i < shape.length; i++) {
        for (int j = 0; j < shape[0].length; j++) {
            System.out.print(shape[i][j]);
        }
        System.out.println();
    }
}
}

```

BAB 3 EKSPERIMEN

Pada bab ini, dilakukan eksperimen menggunakan 9 test case. Setiap test case berupa file input (.txt) digunakan untuk menguji keakuratan solusi serta performa runtime program.

No TC	Input (.txt)	Output
1.	<pre> 5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG </pre>	<pre> BLOCK PUZZLE AUTO SOLVER Input file name located at the /testcase folder: test1.txt ABBCC AABCD EEEDD EEFFF GGGFF Time spent: 202 ms # of cases reviewed: 105178 Would you like to save the solution? (yes/no) yes Enter filename: sol1.txt Solution saved as sol1.txt in the /test folder. </pre>

	Contoh input dari spek	
2.	 <p>Contoh input sederhana buatan penulis</p>	
3.	 <p>Contoh input custom dari spek</p>	

4.	 <p><i>Contoh input salah buatan penulis</i></p>	<pre> BLOCK PUZZLE AUTO SOLVER Input file name located at the /testcase folder: test4.txt No solution found. </pre>
5.	 <p><i>Contoh input block salah buatan penulis</i></p>	<pre> BLOCK PUZZLE AUTO SOLVER Input file name located at the /testcase folder: test5.txt The number of blocks has mismatched the input. </pre>
6.	 <p><i>Contoh input salah buatan penulis</i></p>	<pre> BLOCK PUZZLE AUTO SOLVER Input file name located at the /testcase folder: test6.txt No solution found. </pre>

7.	<pre> 6 6 5 DEFAULT A AAA AAA B BBB BBB C CC CC CC D DDD DDD EEEE EEEE </pre> <p><i>Contoh Test Case input kompleks buatan penulis</i></p>	<pre> BLOCK PUZZLE AUTO SOLVER Input file name located at the /testcase folder: test7.txt AAABBB AAABBB CCABEE CCDDEE CCDDEE CCDDEE Time spent: 304 ms # of cases reviewed: 81226 Would you like to save the solution? (yes/no) yes Enter filename: sol7.txt Solution saved as sol7.txt in the /test folder. </pre>
8.	<pre> 6 6 7 DEFAULT AAAA A A A A AAAA B BB C CC DDD DD EE EE FFFF FFFF G </pre> <p><i>Contoh Test Case input aneh yang dirancang oleh penulis</i></p>	<pre> BLOCK PUZZLE AUTO SOLVER Input file name located at the /testcase folder: test8.txt AAAABB ACGABE ACCAEE AAAAED FFFFDD FFFFDD Time spent: 29326 ms # of cases reviewed: 17352152 Would you like to save the solution? (yes/no) yes Enter filename: sol8.txt Solution saved as sol8.txt in the /test folder. </pre>

9.	<pre> 7 7 7 CUSTOM XX...XX XXX.XXX .XXXXX. ..XXX. .XXXXX. XXX.XXX XX...XX AAA AA AAA BB CC CCC CC D F FF GG GGGG H HHH HH </pre> <p><i>Contoh Test Case besar yang dirancang penulis</i></p>	<pre> Input file name located at the /testcase folder: test9.txt BB CC AAA CCC AACCD AACCD AAA AAA HGGGG HHH GGF HH FF Time spent: 2960 ms # of cases reviewed: 1141804 Would you like to save the solution? (yes/no) yes Enter filename: sol9.txt Solution saved as sol9.txt in the /test folder. </pre>
----	--	--

LAMPIRAN

Link Repository

https://github.com/haegenpro/Tucil1_13523109

Checklist

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	✓	