Laporan Tugas Kecil 1

Syahrizal Bani Khairan - 13523063

Untuk kuliah IF2211 Strategi Algoritma

24 Februari 2025

1. Identifikasi Masalah

Permasalahan yang menjadi topik tugas ini adalah penyelesaian permainan puzzle IQ Puzzler Pro. menggunakan metode brute force. Puzzle terdiri dari sejumlah *block* yang harus ditempatkan pada satu *board*. Dengan itu, komponen penting dari *puzzle* adalah

1. **Board** – dimana *block* ditempatkan. Board berupa *lattice* persegi panjang dan sebuah *block* dapat menempati titik-titik pada *lattice* tersebut. *Board* juga dapat memiliki modifikasi berupa *lattice* yang dapat memiliki bolongan atau konfigurasi piramida/tiga dimensi.
2. **Block** – komponen yang diletakkan pada *board*. *Block* memiliki bentuk pipih/datar dan berupa titik-titik yang terhubung dengan jarak yang sama satu sama lain dan pada sudut siku. *Block* tidak bisa menempati suatu posisi di *board* jika ada bagian yang menempati posisi yang sama dengan bagian *block* lain. *Block* dapat ditempatkan dalam orientasi yang telah dirotasi atau dicerminkan.

Puzzle diawali dengan *board* kosong. Kondisi *puzzle* yang terselesaikan adalah ketika kedua kondisi di bawah dipenuhi:

1. Seluruh *block* ditempatkan dengan benar pada *board,* dan
2. *Board* terisi penuh, tidak ada ruang yang kosong.

Dengan definisi permasalah tersebut, dapat dirumuskan algoritma *brute force* untuk menyelesaikan permasalahan.

2. Spesifikasi

A. Algoritma *Brute Force*

Berdasarkan suatu definisi permasalahan, sebuah algoritma *brute force* dapat dirancang. Algoritma yang digunakan menggunakan teknik rekursi dan *backtracking.* Langkah-langkahnya adalah sebagai berikut:

1. Cek apakah *puzzle* telah selesai. Jika *puzzle* telah selesai, lanjut ke poin 2. Selain itu, untuk setiap *block* yang ada, lakukan hal berikut dimulai 1.1.
   1.1. Ambil suatu posisi pada *board* yang belum dicoba untuk *block* ini. Jika semua posisi telah dicoba, kembali ke poin 1.

1.2. Transformasikan *block* ke orientasi yang belum dicoba untuk posisi tertentu. Jika semua orientasi telah dicoba, kembali ke poin 1.1.
1.3. Cek apakah *block* dapat ditempatkan pada posisi dan orientasi yang diberikan. Jika dapat ditempatkan, maka tempatkan, lanjutkan rekursi ke poin 1, lalu jika *puzzle* masih belum terselesaikan lepaskan *block* dan kembali ke poin 1.2. Jika tidak dapat ditempatkan, kembali ke poin 1.2.

2. *Puzzle* telah selesai atau algoritma tidak menemukan solusi yang valid.

Algoritma ini melakukan e*xhaustive search* terhadap semua peletakkan *block* yang valid pada *board*. Pada tiap level rekursi, dilakukan pengecekan apakah kondisi puzzle telah sesuai dengan dua kondisi selesai yang dijelaskan sebelumnya. Demikian algoritma *brute force* untuk menyelesaikan permasalahan. **Dalam implementasinya, tidak ada penggunaan *heuristic*; kode program hanya mengacu pada definisi permasalahan dan algoritma di atas.**

B. Input dan Output

Program membaca konfigurasi *puzzle* dari sebuah file txt. Ketentuan mengenai file ini dapat ditemukan pada dokumen spesifikasi tugas ini. Program dapat menampilkan solusi yang ditemukan dengan menunjukkan *board* yang telah ditempati *block* (ditampilkan berwarna). Program menunjukkan jumlah kasus yang ditinjau program dan waktu eksekusi algoritma penyelesaian *puzzle*. Hasil program juga dapat disimpan pada file txt.

3. Implementasi

A. Skema Program dan Struktur Data

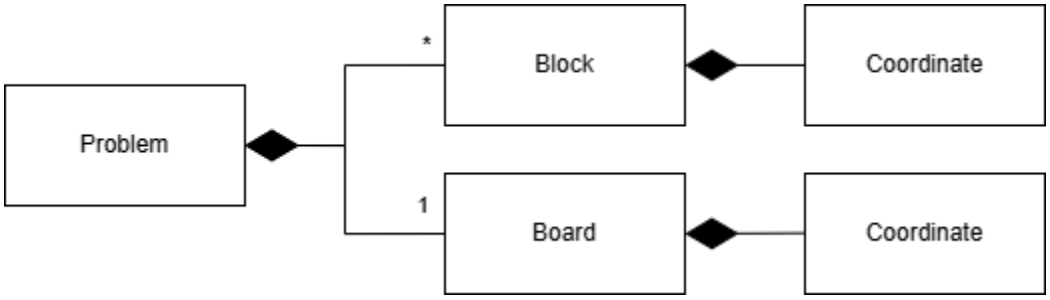
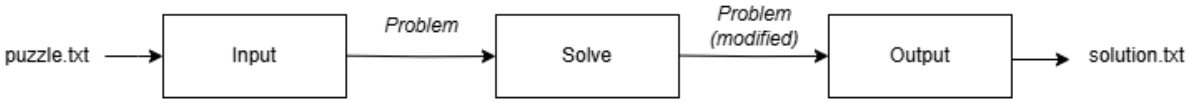


Diagram objek data

Sebuah *puzzle* direpresentasikan sebagai objek *Problem*. Seluruh *state* dan aksi seperti penempatan *block* disimpan dalam objek ini. Sebuah *Problem* terdiri dari satu *Board* dan sejumlah *Block*. *Block* dan *Board* diimplementasikan sebagai sekumpulan *Coordinate* atau objek posisi: sekumpulan koordinat posisi relatif yang menunjukkan bentuk sebuah *Block* dan sebagai sekumpulan posisi pada *Board* yang berkorespondensi dengan indeks array.



Alur program

Terdapat fungsi input yang dapat membaca file txt, melakukan *parsing* dan validasi yang jika berhasil menghasilkan objek *Problem*. Kemudian, *Solver* akan melakukan melakukan penyelesaian dengan memanggil method *Problem*, mengubah *state* objek itu sendiri. *Solver* mengubah *Problem* sehingga terselesaikan jika menemukan solusi. Solusi kemudian ditampilkan melalui Output. File input ditaruh pada direktori ./input dan file output ditaruh pada direktori ./output. Waktu eksekusi *Solver* diukur dengan menghitung selisih sebelum dan sesudah pemanggilan fungsi *Solver.*

```
public class Main {
    public static void main(String[] args) {
        Problem problem = Input.read_file();

        // File input tidak valid
        if (problem == null) {
            return;
        }

        long startTime = System.currentTimeMillis();

        problem = Solve.solve(problem);

        long endTime = System.currentTimeMillis();
        long timeElapsed = endTime - startTime;

        Output.display_solution(problem, timeElapsed);

        return;
    }
}
```

A.1. Problem

```
public class Problem {
    public static final String[] modes = {"DEFAULT", "CUSTOM", "PYRAMID"};

    int width, height, blocksAvailable;
    String mode;

    Board board;

    Block[] blocks;
    boolean[] usedBlock;
```

```
    Coordinate[] blockPositions;

    boolean isSolved;
    int caseTested;

    public Coordinate[] get_possible_coordinates();
    public boolean placeable_block(int block_index, Coordinate coordinate);
    public void place_block(int block_index, Coordinate coordinate);
    public void remove_block(int block_index, Coordinate coordinate);
    public boolean solved();
}
```

Solver akan memanggil fungsi yang ditunjukkan untuk mengubah *state* dan melakukan pencarian solusi. Salah satu atribut usedBlock menyimpan status penempatan *Block* dalam puzzle.

A.2. Board

```
public class Board {
    boolean validBoard;
    char[][] rectangularBoard;
    char[][][] pyramidBoard;
    int width, height, altitude;
    int emptySpace;
    Coordinate[] possibleCoordinates;

    public int get_empty_space();
    public Coordinate[] get_possible_coordinates();

    public Coordinate add_coordinate(Coordinate origin, Coordinate add);

    public boolean placeable_block(Block block, Coordinate coord);
    public void place_block(Block block, Coordinate coord);
    public void remove_block(Block block, Coordinate coord)
```

Bentuk fisik *Board* diabstraksi dari *Solver*. *Board* terdiri dari sekumpulan *Coordinate* di mana *Block* dapat diletakkan. Selain itu, *Board* memiliki atribut integer yang menghitung jumlah ruang kosong yang ada di *Board* dan nilainya diupdate setiap kali *Block* ditempatkan atau dilepas.

A.3. Block

```
public class Block {
    char name;
    Coordinate[] shape;

    public void reflect_i();
    public void rotate_90_cw_k(int n);
```

```
    public void raise();
    public void flatten();
    public void reflect_k();
```

Block pada dasarnya adalah sekumpulan *Coordinate* yang menunjukkan posisi relatif dari setiap titik yang membentuk *Block* itu sendiri. Salah satu titik pada *Block* dijadikan sebagai ‘origin’ dengan koordinat (0, 0, 0) dan relatif dari titik inilah semua koordinat diukur. *Block* dapat mengalami transformasi, berupa fungsi transformasi geometri sederhana, yang mengubah tiap koordinat untuk mendapatkan orientasi baru pada ruang papan.

B. Implementasi Brute Force

Sebuah *Problem* dapat diselesaikan dengan memberikannya ke fungsi Solve:

```
public static Problem solve(Problem problem) {
        if (problem.mode.equals("DEFAULT") || problem.mode.equals("CUSTOM")) {
            Solve.recurse_2D(problem, 0);
        } else if (problem.mode.equals("PYRAMID")) {
            Solve.recurse_3D(problem, 0);
        } else {
            System.out.println("Invalid mode");
            return null;
        }
        return problem;
    }
```

Untuk modifikasi “CUSTOM”, langkah brute force tidak berubah, karena abstraksi pada *Board* langsung memberikan list posisi legal. Tetapi pada modifikasi “PYRAMID”, diperlukan sedikit perubahan karena bertambahnya kemungkinan orientasi *Block*. Dalam rekursi, akan berkali-kali memanggil fungsi Problem.solved() untuk mengecek kondisi penyelesaian. Fungsi tersebut didefinisikan sebagai berikut:

```
public class Problem {
    Board board;

    Block[] blocks;
    boolean[] usedBlock;
    Coordinate[] blockPositions;

    boolean isSolved;
    int caseTested;

    public boolean solved() {
        if (this.isSolved) {
            return true;    // Already passed all solve condition
        }
```

```
        /* Solve conditions */
        // Completely filled
        if (this.board.get_empty_space() != 0) {
            return false;
        }

        // All blocks used
        for (int i=0;i<this.blocksAvailable;i++) {
            if (!this.usedBlock[i]) {
                return false;
            }
        }

        this.isSolved = true;
        return true; // Yay!
    }
```

Pada awalnya, atribut isSolved diset ke False. Pertama, fungsi di atas akan mengecek apakah *Board* telah diisi sepenuhnya, sesuai kondisi kedua penyelesaian. Lalu, fungsi akan mengecek apakah semua *Block* telah ditempatkan, sesuai kondisi pertama penyelesaian(penempatan *Block* hanya bisa dilakukan jika posisi dan orientasi valid). Jika kedua kondisi telah dipenuhi, isSolved akan diset ke True dan puzzle dinyatakan memiliki solusi.

Fungsi rekursi untuk kasus 2D adalah berikut:

```
    public static void recurse_2D(Problem problem, int blockIndex) {
        if (blockIndex >= problem.blocksAvailable || problem.solved()) {
            return;
        }

        Block currentBlock = problem.blocks[blockIndex];
        Coordinate[] possibleCoordinates = problem.get_possible_coordinates();
        // Place block on each possible position
        for (int i=0;i<possibleCoordinates.length;i++) {
            Coordinate coord = possibleCoordinates[i];

            // recurse_2D_branch place the block, recurse, remove, then rotate
the block
            // recurse_2D_branch does not do anything if the problem is solved

            // Not flipped. All rotations
            Solve.recurse_2D_branch(problem, blockIndex, coord);
            Solve.recurse_2D_branch(problem, blockIndex, coord);
            Solve.recurse_2D_branch(problem, blockIndex, coord);
            Solve.recurse_2D_branch(problem, blockIndex, coord);
```

```
            // Horizontally flipped. All rotations
            if (!problem.solved()) {
                currentBlock.reflect_i();
                Solve.recurse_2D_branch(problem, blockIndex, coord);
                Solve.recurse_2D_branch(problem, blockIndex, coord);
                Solve.recurse_2D_branch(problem, blockIndex, coord);
                Solve.recurse_2D_branch(problem, blockIndex, coord);

                if (!problem.solved()) {
                    currentBlock.reflect_i();
                }
            }
        }
    }

    public static void recurse_2D_branch(Problem problem, int blockIndex,
Coordinate coord) {
        if (!problem.solved()) {
            problem.caseTested++;
            if (problem.placeable_block(blockIndex, coord)) {
                problem.place_block(blockIndex, coord);

                recurse_2D(problem, blockIndex+1);

                if (problem.solved()) {
                    return;
                }

                problem.remove_block(blockIndex, coord);
            }

            problem.blocks[blockIndex].rotate_90_cw_k(1);
        }
    }
```

Pada recurse_2D, terdapat enumerasi posisi *Board* dan orientasi *Block*. Fungsi ini memanggil fungsi recurse_2D_branch untuk mencoba menempatkan *Block* dan melanjutkan rekursi lebih dalam. Fungsi rekursi_2D_branch juga dapat melakukan transformasi *Block* di akhir. Untuk kasus 2D, ada 8 orientasi unik untuk benda datar umum. Orientasi ini didapatkan dengan 4 kali melakukan rotasi 90 derajat terhadap sumbu-z, dan 4 kali lagi dengan *Block* yang direfleksi terlebih dahulu. Jumlah kasus yang diuji bertambah satu untuk setiap kali pengujian peletakan *Block*. Skenario terbaik adalah jumlah kasus teruji adalah sama dengan jumlah *Block* yang ada.

Pada kasus 3D, pendekatannya sama.

```
    public static void recurse_3D(Problem problem, int blockIndex) {
```

```
        if (blockIndex >= problem.blocksAvailable || problem.solved()) {
            return;
        }

        Block currentBlock = problem.blocks[blockIndex];
        Coordinate[] possibleCoordinates = problem.get_possible_coordinates();
        // Place block on each possible position
        for (int i=0;i<possibleCoordinates.length;i++) {
            Coordinate coord = possibleCoordinates[i];

            /* Planar */
            // Not flipped. All rotations
            Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
            Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
            Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
            Solve.recurse_3D_branch(problem, blockIndex, coord, 0);

            // Horizontally flipped. All rotations
            if (!problem.solved()) {
                currentBlock.reflect_i();
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
            }

            /* 3D */
            if (!problem.solved()) {
                currentBlock.reflect_i();
                currentBlock.raise();

                // Not flipped. All rotations about the vertical
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);

                // Vertically flipped. All rotations about the vertical
                if (!problem.solved()) {
                    currentBlock.reflect_k();
                    Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                    Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                    Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                    Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                }
```

```
                if (!problem.solved()) {
                    currentBlock.reflect_k();
                    currentBlock.flatten();
                }
            }

            // Flipped along i-j
            if (!problem.solved()) {
                currentBlock.reflect_i_j();
                currentBlock.raise();

                // Not flipped. All rotations about the vertical
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                Solve.recurse_3D_branch(problem, blockIndex, coord, 0);

                // Vertically flipped. All rotations about the vertical
                if (!problem.solved()) {
                    currentBlock.reflect_k();
                    Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                    Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                    Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                    Solve.recurse_3D_branch(problem, blockIndex, coord, 0);
                }

                if (!problem.solved()) {
                    currentBlock.reflect_k();
                    currentBlock.flatten();
                    currentBlock.reflect_i_j();
                }
            }


        }
    }

    public static void recurse_3D_branch(Problem problem, int blockIndex,
Coordinate coord, int transformMode) {
        if (!problem.solved()) {
            problem.caseTested++;

            if (problem.placeable_block(blockIndex, coord)) {
                problem.place_block(blockIndex, coord);

                recurse_3D(problem, blockIndex+1);
```

```
                if (problem.solved()) {
                    return;
                }

                problem.remove_block(blockIndex, coord);
            }

            if (transformMode == 0) {
                problem.blocks[blockIndex].rotate_90_cw_k(1);
            } else if (transformMode == 2) {
                problem.blocks[blockIndex].reflect_i();
            }
        }
    }
```

Pada modifikasi “PYRAMID”, mungkinnya komponen vertikal menghasilkan jumlah orientasi yang lebih banyak. *Block* yang datar dijadikan vertikal dengan method raise(). Dengan itu, *Block* yang merupakan objek datar memiliki normal yang paralel dengan vektor i-j. Program akan menguji 16 orientasi baru pada kasus ini, yaitu dengan melakukan 4 rotasi 90 derajat terhadap sumbu-z, dan 4 lagi rotasi setelah direfleksi secara vertikal. Transformasi tadi diulangi lagi dengan refleksi paralel dengan vektor i-j. *Block* juga masih dapat menempati *Board* dengan posisi datar selain orientasi 3 dimensi.

4. Hasil Uji

Tangkapan layar juga tersedia di direktori test pada repositori

4.1. “DEFAULT”

```
Enter the file name with extension : sample.txt

Solusi ditemukan!
AABBG
CADBG
CCDDG
EEEFF
EEFFF

Waktu pencarian: 1 ms

Jumlah kasus yang ditinjau: 488
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: sample_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : test1.txt

Solusi ditemukan!
AEBB
AABB
ACBD
ACDD
ACFF

Waktu pencarian: 0 ms

Jumlah kasus yang ditinjau: 344
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: test1_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : test2.txt

Solusi ditemukan!
AAA
BBB
CCC

Waktu pencarian: 0 ms

Jumlah kasus yang ditinjau: 75
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: test2_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : test3.txt

Solusi ditemukan!
AAAC
ABAC
AAAC
DDDD

Waktu pencarian: 0 ms

Jumlah kasus yang ditinjau: 13481
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: test3_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : test4.txt

Solusi ditemukan!
AAAC
ABAC
AAAC
DDDD

Waktu pencarian: 0 ms

Jumlah kasus yang ditinjau: 169
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: test4_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : test5.txt

Solusi ditemukan!
AAAC
ABAB
ABAB
DBBB

Waktu pencarian: 1 ms

Jumlah kasus yang ditinjau: 238
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: test5_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : test6.txt

Solusi ditemukan!
ABCDEFGHIJKLM
NOPQRSTUVWXYZ

Waktu pencarian: 1 ms

Jumlah kasus yang ditinjau: 2626
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: test6_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : test7.txt
WARNING: The blocks provided do not have the total space that ca
n completely occupy and fit the whole board. No solution is poss
ible
Attempt to solve the problem anyway...

Tidak ada solusi yang ditemukan

Waktu pencarian: 112 ms

Jumlah kasus yang ditinjau: 2318600
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: test7_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : test8.txt
WARNING: The blocks provided do not have the total space that ca
n completely occupy and fit the whole board. No solution is poss
ible
Attempt to solve the problem anyway...

Tidak ada solusi yang ditemukan

Waktu pencarian: 0 ms

Jumlah kasus yang ditinjau: 72
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: test8_solution.txt
Solusi berhasil disimpan!
```

4.2. “CUSTOM”

```
Enter the file name with extension : custom_sample.txt

Solusi ditemukan!
   A
 AAABB
CCCCBBB
 CDEEE
   E

Waktu pencarian: 11 ms

Jumlah kasus yang ditinjau: 31649
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: custom_sample_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : custom1.txt

Solusi ditemukan!
 A  B
GAFBBB
GA  BC
EE  CC
EEDDDD
 E  D

Waktu pencarian: 6 ms

Jumlah kasus yang ditinjau: 32030
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: custom1_solution.txt
Solusi berhasil disimpan!
```

4.1. “PYRAMID”

```
Enter the file name with extension : pyramid_sample.txt

Solusi ditemukan!
D
ED
EE
AAD
BAF
BCC

Waktu pencarian: 0 ms

Jumlah kasus yang ditinjau: 699
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: pyramid_sample_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : pyramid1.txt

Solusi ditemukan!
A
AB
BB

Waktu pencarian: 0 ms

Jumlah kasus yang ditinjau: 565
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: pyramid1_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : pyramid2.txt

Solusi ditemukan!
B
CC
CB
AAA
ABA
AAD

Waktu pencarian: 0 ms

Jumlah kasus yang ditinjau: 522
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: pyramid2_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : pyramid3.txt

Solusi ditemukan!
D
CC
CD
AAA
ABB
ABD

Waktu pencarian: 0 ms

Jumlah kasus yang ditinjau: 525
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: pyramid3_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : pyramid4.txt

Tidak ada solusi yang ditemukan

Waktu pencarian: 4161 ms

Jumlah kasus yang ditinjau: 297130320
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: pyramid4_solution.txt
Solusi berhasil disimpan!
```

```
Enter the file name with extension : pyramid5.txt

Solusi ditemukan!
B
CB
BD
AAB
AAA
BAA

Waktu pencarian: 2 ms

Jumlah kasus yang ditinjau: 836
Apakah anda ingin menyimpan solusi? [Y/N] Y
Masukkan nama file: pyramid5_solution.txt
Solusi berhasil disimpan!
```

5. Lampiran

Link spesifikasi tugas: Spesifikasi Tugas Kecil 1 Stima 2024/2025

Link repository: https://github.com/rizalkhairan/Tucil1_13523063

Checklist pengerjaan tugas:

No	Poin	Ya	Tidak

1	Program berhasil dikompilasi tanpa kesalahan	v	
2	Program berhasil dijalankan	v	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	v	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	v	
5	Program memiliki *Graphical User Interface* (GUI)		v
6	Program dapat menyimpan solusi dalam bentuk file gambar		v
7	Program dapat menyelesaikan kasus konfigurasi *custom*	v	
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)	v	
9	Program dibuat oleh saya sendiri	v	