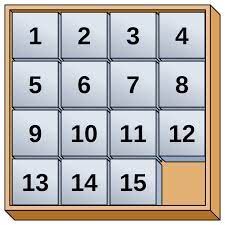
**LAPORAN TUGAS KECIL IF2211 STRATEGI ALGORITMA SEMESTER II TAHUN 2021/2022**

**IMPLEMENTASI CONVEX HULL UNTUK VISUALISASI TES LINEAR SEPARABILITY DATASET DENGAN ALGORITMA DIVIDE AND CONQUER**



**Disusun oleh:**

**13520156 Dimas Faidh Muzaki**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

# *Cara Kerja*

# *Screen-shot*

# *Checklist*

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi | ✓ |  |
| 1. Program berhasil *running* | ✓ |  |
| 1. Program dapat menerima input dan menuliskan output | ✓ |  |
| 1. Luaran sudah benar untuk semua data | ✓ |  |
| 1. Bonus dibuat |  | ✓ |

# *Kode Program*

/\*\*

 \* Main.java

 \*/

public class Main  {

    public static void main(String[] args) {

        Board b = null;

        try {

            b = Board.fromFile(args[0]);

        } catch (Exception e) {

            if (e instanceof NoSuchFieldException) {

                System.out.println("File tidak ditemukan, membuat board random...");

            }

            b = Board.random();

        }

        System.out.println();

        System.out.println("Board puzzle berhasil dibuat");

        b.show();

        int kurang = 0;

        System.out.println();

        System.out.println(" \_\_\_\_\_ \_\_\_\_\_\_\_\_\_");

        System.out.println("|  i  |Kurang(i)|");

        System.out.println("|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|");

        for (int i = 1; i <= 16; i++) {

            int kurangi = PuzzleSolver.kurang(b, i);

            kurang += kurangi;

            if (i < 10) {

                System.out.print("|  " + i + "  |");

            } else {

                System.out.print("| " + i + "  |");

            }

            if (kurangi < 10) {

                System.out.println("    " + kurangi + "    |");

            } else {

                System.out.println("   " + kurangi + "   |");

            }

        }

        System.out.println("| tot |    "+kurang+"   |");

        System.out.println("|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|");

        System.out.println("|  X  |    " + PuzzleSolver.xBesar(b) + "    |");

        System.out.println("|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|+");

        System.out.println("|       "+(kurang + PuzzleSolver.xBesar(b))+"      |");

        System.out.println("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|");

        if (!b.isSolvable()) {

            System.out.println("Status tujuan tidak dapat dicapai.");

        } else{

            long startTime = System.nanoTime();

            // ArrayList<Board> solution =

            Pair<Board,Integer> result = PuzzleSolver.solve(b);

            long stopTime = System.nanoTime();

            // 1 second = 1\_000\_000\_000 nano seconds

            double elapsedTimeInSecond = (double) (stopTime - startTime) / 1\_000\_000\_000;

            b.showSolution(result.getKey().getTail());

            System.out.println("Waktu eksekusi: " + elapsedTimeInSecond + " seconds");

            System.out.println("Jumlah node dibangkitkan: " + result.getValue());

        }

    }

}

/\*\*

 \* Board.java

 \*/

import java.io.BufferedReader;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.stream.Collectors;

public class Board {

    private ArrayList<String> tail;

    private int f;

    private int c;

    private int g;

    private int N;

    private int empty\_x;

    private int empty\_y;

    private int[][] board;

    public Board() {

        super();

        this.tail = new ArrayList<String>();

        this.f = 0;

        this.N = 4;

        this.empty\_x = 0;

        this.empty\_y = 0;

        this.c = 0;

        this.g = 0;

        this.board = new int[4][4];

    }

    public Board(Board inputBoard) {

        super();

        this.tail = new ArrayList<String>();

        this.N = inputBoard.N;

        this.f = inputBoard.f;

        this.c = inputBoard.c;

        this.g = inputBoard.g;

        this.empty\_x = inputBoard.empty\_x;

        this.empty\_y = inputBoard.empty\_y;

        this.board = new int[this.N][this.N];

        for (int i = 0; i < this.N; i++) {

            for (int j = 0; j < this.N; j++) {

                this.board[i][j] = inputBoard.board[i][j];

            }

        }

    }

    public void addTail(String input) {

        this.tail.add(input);

    }

    public ArrayList<String> getTail() {

        return tail;

    }

    public int getC() {

        return c;

    }

    public int getG() {

        return g;

    }

    public int getElmt(int i, int j){

        return this.board[i][j];

    }

    public int getN() {

        return N;

    }

    public int getF() {

        return f;

    }

    public int getEmpty\_x() {

        return empty\_x;

    }

    public int getEmpty\_y() {

        return empty\_y;

    }

    public void setF(int f) {

        this.f = f;

    }

    public void setN(int n) {

        N = n;

    }

    public void setEmpty\_x(int empty\_x) {

        this.empty\_x = empty\_x;

    }

    public void setEmpty\_y(int empty\_y) {

        this.empty\_y = empty\_y;

    }

    public void setElmt(int i, int j, int X){

        this.board[i][j] = X;

    }

    private void swap(int i, int j, int k, int l) {

        int temp = this.board[i][j];

        this.board[i][j] = this.board[k][l];

        this.board[k][l] = temp;

    }

    public static Board fromFile(String fileName) {

        Board A = new Board();

        String currDir = System.getProperty("user.dir");

        fileName = currDir + "\\config\\" + fileName;

        List<String> list = new ArrayList<>();

        try (BufferedReader br = Files.newBufferedReader(Paths.get(fileName))) {

            //br returns as stream and convert it into a List

            list = br.lines().collect(Collectors.toList());

            int i = 0;

            for (String string : list) {

                int j = 0;

                String[] splited = string.split("\\s+");

                for (String string2 : splited) {

                    A.board[i][j] = Integer.parseInt(string2);

                    if (Integer.parseInt(string2) == 16) {

                        A.empty\_x = i;

                        A.empty\_y = j;

                    }

                    j++;

                }

                i++;

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

        return A;

    }

    public static Board random() {

        Board A = new Board();

        ArrayList<Integer> l = new ArrayList<Integer>();

        Random rand = new Random();

        for (int i = 1; i<=16;i++){

            l.add(i);

        }

        for (int i = 0; i < 4; i++)

            for (int j = 0; j < 4; j++){

                    int randomIndex = rand.nextInt(l.size());

                    int randomElement = l.remove(randomIndex);

                    if (randomElement == 16){

                        A.empty\_x = i;

                        A.empty\_y = j;

                    }

                    A.board[i][j] = randomElement;

            }

        return A;

    }

    public void show() {

        System.out.println(" \_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_");

        for (int i = 0; i < N; i++) {

            System.out.print("|");

            for (int j = 0; j < N; j++) {

                int x = this.board[i][j];

                if (x == 16) {

                    System.out.print("    |");

                } else {

                    if (x == (i\*4)+j+1){

                        if (x < 10) {

                            System.out.print("\*");

                            System.out.print(" " + x);

                            System.out.print("\*|");

                        } else{

                            System.out.print("\*");

                            System.out.print(x);

                            System.out.print("\*|");

                        }

                    } else {

                        if (x < 10) {

                            System.out.print(" ");

                            System.out.print(" " + x);

                            System.out.print(" |");

                        } else{

                            System.out.print(" ");

                            System.out.print(x);

                            System.out.print(" |");

                        }

                    }

                }

            }

            System.out.println();

            System.out.println("|\_\_\_\_|\_\_\_\_|\_\_\_\_|\_\_\_\_|");

        }

    }

    public void updateG() {

        this.g = PuzzleSolver.gTopi(this);

    }

    public void updateC() {

        this.updateG();

        this.c = this.f + this.g;

    }

    public void showSolution(ArrayList<String> list) {

        for (String string : list) {

            if (string.equals("up")) {

                this.up();

                this.show();

            } else if (string.equals("down")) {

                this.down();

                this.show();

            } else if (string.equals("left")) {

                this.left();

                this.show();

            } else if (string.equals("right")) {

                this.right();

                this.show();

            }

        }

    }

    public void up() {

        swap(empty\_x, empty\_y, empty\_x-1, empty\_y);

        empty\_x--;

        this.updateC();

    }

    public void down() {

        swap(empty\_x, empty\_y, empty\_x+1, empty\_y);

        empty\_x++;

        this.updateC();

    }

    public void left() {

        swap(empty\_x, empty\_y, empty\_x, empty\_y-1);

        empty\_y--;

        this.updateC();

    }

    public void right() {

        swap(empty\_x, empty\_y, empty\_x, empty\_y+1);

        empty\_y++;

        this.updateC();

    }

    public boolean isSolvable() {

        int kurang = 0;

        for (int i = 1; i <= 16; i++) {

            int kurangi = PuzzleSolver.kurang(this, i);

            kurang += kurangi;

        }

        kurang += PuzzleSolver.xBesar(this);

        return (kurang % 2 == 0);

    }

    public boolean equals(Board b){

        for (int i = 0; i < N; i++) {

            for (int j = 0; j < N; j++) {

                if (this.board[i][j] != b.board[i][j]){

                    return false;

                }

            }

        }

        return true;

    }

}

/\*\*

 \* Pair.java

 \*/

public class Pair<T,B> {

    T key;

    B value;

    public Pair() {

        super();

        this.key = null;

        this.value = null;

    }

    public Pair(T key, B value) {

        super();

        this.key = key;

        this.value = value;

    }

    public T getKey() {

        return key;

    }

    public B getValue() {

        return value;

    }

}

/\*\*

 \* PuzzleSolver.java

 \*/

import java.util.ArrayList;

public class PuzzleSolver {

    public static int kurang(Board b, int x) {

        boolean start = false;

        int counter = 0;

        for (int i = 0; i < b.getN(); i++) {

            for (int j = 0; j < b.getN(); j++) {

                if (b.getElmt(i, j) == x){

                    start = true;

                }

                if(start && b.getElmt(i, j) < x && b.getElmt(i, j) != 0){

                    counter++;

                }

            }

        }

        return counter;

    }

    public static int xBesar(Board b) {

        int i = 0;

        int j = 0;

        int ires = 0;

        int jres = 0;

        for (i = 0; i < b.getN(); i++) {

            for (j = 0; j < b.getN(); j++) {

                if (b.getElmt(i, j) == 16){

                    ires = i;

                    jres = j;

                }

            }

        }

        return (((ires + jres) % 2) == 0 ? 0: 1);

    }

    public static int gTopi(Board b) {

        int counter = 0;

        for (int i = 0; i < b.getN(); i++) {

            for (int j = 0; j < b.getN(); j++) {

                // System.out.println(Integer.toString(b.getElmt(i, j)) + " == " +  Integer.toString((i\*4)+j+1));

                if(b.getElmt(i, j)!= (i\*4)+j+1 && b.getElmt(i, j)!= 16)

                    counter++;

            }

        }

        return counter;

    }

    public static int fTopi(Board b) {

        return b.getF();

    }

    public static int cTopi(Board b) {

        return gTopi(b) + fTopi(b);

    }

    public static boolean isSolution(Board b) {

        return gTopi(b) == 0;

    }

    public static int minFour(int one, int two, int three, int four) {

        return Math.min(one, Math.min(two, Math.min(three, four)));

    }

    public static int findMin(ArrayList<Pair<Board, String>> list) {

        int min = Integer.MAX\_VALUE;

        int idx = 0;

        int i = 0;

        for (Pair<Board, String> pair : list) {

            if (pair.getKey().getC() < min) {

                min = pair.getKey().getC();

                idx = i;

            }

            i++;

        }

        return idx;

    }

    public static boolean isBoardEqual(Pair<Board, String> b1, Pair<Board, String> b2) {

        return b1.getKey().equals(b2.getKey());

    }

    public static boolean has(ArrayList<Pair<Board, String>> list, Pair<Board, String> b) {

        for (Pair<Board, String> pair : list) {

            if (isBoardEqual(pair, b)) {

                return true;

            }

        }

        return false;

    }

    public static Pair<Board, Integer> solve(Board b){

        ArrayList<Pair<Board,String>> boardList = new ArrayList<Pair<Board,String>>();

        Board currBoard = new Board(b);

        boolean end = false;

        Pair<Board, String> currPair = new Pair<Board, String>(currBoard, "");

        int nodeCount = 1;

        while (!end && currPair.getKey().isSolvable()) {

            ArrayList<Pair<Board,String>> localBoardList = new ArrayList<Pair<Board,String>>();

            Pair<Board, String> upPair;

            Pair<Board, String> rightPair;

            Pair<Board, String> leftPair;

            Pair<Board, String> downPair;

            if (!(currPair.getKey().getEmpty\_x() == 0 || currPair.getValue().equals("down"))) {

                upPair = new Pair<Board, String>(new Board(currPair.getKey()), "up");

                upPair.getKey().setF(currPair.getKey().getF() + 1);

                for (String path : currPair.getKey().getTail()) {

                    upPair.getKey().addTail(path);

                }

                upPair.getKey().addTail("up");

                upPair.getKey().up();

                    localBoardList.add(upPair);

                nodeCount++;

                // if (!has(boardList, upPair)) {

                //     localBoardList.add(upPair);

                // }

            }

            if (!(currPair.getKey().getEmpty\_x() == currPair.getKey().getN() - 1 || currPair.getValue().equals("up"))) {

                downPair = new Pair<Board, String>(new Board(currPair.getKey()), "down");

                downPair.getKey().setF(currPair.getKey().getF() + 1);

                for (String path : currPair.getKey().getTail()) {

                    downPair.getKey().addTail(path);

                }

                downPair.getKey().addTail("down");

                downPair.getKey().down();

                localBoardList.add(downPair);

                nodeCount++;

                // if (!has(boardList, downPair)) {

                //     localBoardList.add(downPair);

                // }

            }

            if (!(currPair.getKey().getEmpty\_y() == 0 || currPair.getValue().equals("right"))) {

                leftPair = new Pair<Board, String>(new Board(currPair.getKey()), "left");

                leftPair.getKey().setF(currPair.getKey().getF() + 1);

                for (String path : currPair.getKey().getTail()) {

                    leftPair.getKey().addTail(path);

                }

                leftPair.getKey().addTail("left");

                leftPair.getKey().left();

                    localBoardList.add(leftPair);

                nodeCount++;

                // if (!has(boardList, leftPair)) {

                //     localBoardList.add(leftPair);

                // }

            }

            if (!(currPair.getKey().getEmpty\_y() == currPair.getKey().getN() - 1 || currPair.getValue().equals("left"))) {

                rightPair = new Pair<Board, String>(new Board(currPair.getKey()), "right");

                rightPair.getKey().setF(currPair.getKey().getF() + 1);

                for (String path : currPair.getKey().getTail()) {

                    rightPair.getKey().addTail(path);

                }

                rightPair.getKey().addTail("right");

                rightPair.getKey().right();

                localBoardList.add(rightPair);

                nodeCount++;

                // if (!has(boardList, rightPair)) {

                //     localBoardList.add(rightPair);

                // }

            }

            for (Pair<Board,String> pair : localBoardList) {

                boardList.add(pair);

            }

            int iMin = PuzzleSolver.findMin(boardList);

            currPair = boardList.get(iMin);

            boardList.remove(iMin);

            localBoardList.clear();

            if (isSolution(currPair.getKey())) {

                ArrayList<Pair<Board,String>> fooListCopy = new ArrayList<Pair<Board,String>>();

                for (Pair<Board,String> pair : boardList) {

                    if (pair.getKey().getC() < currPair.getKey().getC()) {

                        fooListCopy.add(pair);

                    }

                }

                boardList = fooListCopy;

                if(boardList.isEmpty()){

                    end = true;

                }

                else{

                    currPair = boardList.get(PuzzleSolver.findMin(boardList));

                }

            }

        }

        return new Pair<Board,Integer>(currPair.getKey(), nodeCount);

    }

}

# *Test Case*

# *Link Kode Program*