Laporan Tugas Kecil 3 “Fifteen Puzzle Solver”

IF2211 Strategi Algoritma

Disusun oleh:

Haidar Ihzaulhaq 13520150

Sekolah Tinggi Elektro dan Informatika

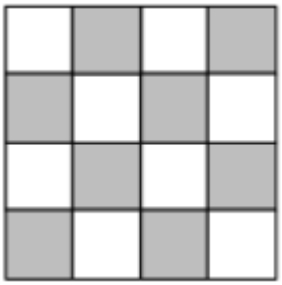
Institut Teknologi Bandung

2021/2022

1. **Cara Kerja Algoritma Branch & Bound**

   Program ini adalah program untuk menyelesaikan permainan Puzzle-15, yaitu sebuah permainan puzzle 16 kotak yang diisi dengan angka 1-15 secara acak dan satu kotak yang kosong. Pemain diharuskan untuk menggerakkan kotak yang berisi angka ke kotak kosong hingga susunan angka menjadi urut dari 1-15 secara horizontal. Permainan ini dapat diselesaikan salah satunya dengan algoritma Branch and Bound. Secara singkat, cara kerja algoritma ini dalam penyelesaian permainan Puzzle-15 adalah sebagai berikut:

   a) Pertama, susunan acak angka pada puzzle harus dihitung terlebih dahulu apakah bisa untuk diselesaikan atau tidak. Caranya adalah dengan melakukan perhitungan nilai Kurang(i) dan kemudian menjumlahkan semua nilai Kurang(i) dari 1 sampai 16 (kotak kosong dianggap bernilai 16). Nilai Kurang(i) adalah banyaknya angka j yang nilainya kurang dari i, namun memiliki posisi yang lebih besar dari i (posisi[j] > posisi[i]). Kemudian, setelah semua nilai Kurang(i) dihitung dan dijumlahkan, selanjutnya adalah menambahkannya dengan nilai X. Nilai X memiliki ketentuan bernilai satu jika kotak kosong berada pada posisi berwarna gelap dan bernilai nol jika berada pada posisi terang pada gambar dibawah.

      Jika ternyata hasil penjumlahan bernilai genap, maka puzzle dapat diselesaikan. Namun jika bernilai ganjil, maka Puzzle tidak dapat diselesaikan.

   b) Jika dapat diselesaikan, maka baru akan masuk pada Algoritma Branch & Bound. Pertama, state awal atau kondisi awal dapat dianggap sebagai simpul akar. Nantinya, setiap simpul akan memiliki beberapa nilai, yaitu:
      - c(i) atau cost(i) = nilai ongkos untuk setiap simpul i
      - f(i) = nilai ongkos untuk mencapai simpul i dari akar
      - g(i) = nilai ongkos mencapai simpul tujuan dari simpul i.
      - dengan c(i) = f(i) + g(i).
      - f(i) dalam hal ini adalah kedalaman simpul i.
      - g(i) adalah jumlah ubin tidak kosong tidak terdapat pada susunan akhir.

   c) Kemudian, dari simpul akar, akan dicek, jika ternyata simpul akar sudah mencapai tujuan yang ingin dicapai (mengurutkan semua angka), maka program berhenti. Jika tidak, maka dari simpul tersebut akan dibangkitkan simpul lainnya. Pembangkitan ini dilakukan dengan menggeser kotak kosong ke arah atas, bawah, kanan, atau kiri. Pembangkitan tidak boleh dilakukan berlawanan arah dengan arah sebelumnya (misal sebelumnya diambil kanan, maka pada pembangkitan sekarang tidak boleh ambil kiri, karena akan sama saja Kembali ke posisi sebelumnya).

   d) Setelah dilakukan pembangkitan, maka dicari simpul yang memiliki cost(i) paling kecil dari semua simpul yang hidup (simpul yang melakukan pembangkitan dianggap sudah mati/dimatikan). Pemrograman untuk kasus ini dapat memanfaatkan priority queue sehingga simpul yang memiliki cost terkecil akan otomatis berada pada atas antrian.

e) Pengecekan akan dilakukan pada simpul yang sekarang (urutan pertama dari priority queue) apakah telah mencapai tujuan yang diinginkan, jika tidak, maka lakukan langkah pembangkitan seperti poin (c) hingga (e) hingga tujuan yang diinginkan tercapai.

2. Source Code Program

a) Class Matrix Puzzle

```
public class Matrix {
    //Class untuk matrix puzzle yang akan dibuat
    private ArrayList<String> step;
    private int[][] matrix;
    private int depth;

    public Matrix() {
        //randomly generate a matrix
        step = new ArrayList<String>();
        this.matrix = new int[4][4];
        ArrayList<Integer> mylist = new ArrayList<Integer>();
       for (int i = 0; i < 16; i++) {
           mylist.add(i);
       }
       Collections.shuffle(mylist);
           for (int i = 0; i < 4; i++) {
                  for (int j = 0; j < 4; j++) {
                         matrix[i][j] = mylist.get(i*4+j);
                  }
           }
           this.depth = 0;
    }

    public Matrix(File file) throws FileNotFoundException {
        //read a matrix from a file
        step = new ArrayList<String>();
        this.depth = 0;
        this.matrix = new int[4][4];
        //read from txt file
        int row = 0;
        File myObj = file;
        Scanner myReader = new Scanner(myObj);
        while (myReader.hasNextLine()) {
               String data = myReader.nextLine();
               String[] data_split = data.split(" ");
               for (int i = 0; i < 4; i++) {
                      matrix[row][i] = Integer.parseInt(data_split[i]);
               }
               row++;
        }
        myReader.close();
    }

    public Matrix(Matrix m) {
        //copy a matrix
        step = new ArrayList<String>();
        this.depth = m.depth;
        this.matrix = new int[4][4];
        for (int i = 0; i < 4; i++) {
               for (int j = 0; j < 4; j++) {
                      this.matrix[i][j] = m.matrix[i][j];
               }
        }
        for (int i = 0; i < m.step.size(); i++) {
               this.step.add(m.step.get(i));
```

```
		}
	}

	public void setElement(int i, int j, int value) {
		//mengubah nilai elemen pada matrix[i][j] menjadi value
		matrix[i][j] = value;
	}

	public void add_step(String step) {
		//menambahkan langkah yang telah dilakukan
		this.step.add(step);
	}

	public int get_element(int i, int j) {
		//Mengembalikan nilai elemen matrix[i][j]
		return this.matrix[i][j];
	}
	public Coordinate getEmptyPos() {
		//Mengembalikan koordinat kosong (dalam hal ini angka 0)
		Coordinate zero = new Coordinate();
		for (int i = 0; i < 4; i++) {
			for (int j = 0; j < 4; j++) {
				if (matrix[i][j] == 0) {
					zero.setX(i);
					zero.setY(j);
					return zero;
				}
			}
		}
		return zero;
	}

	public void printMatrix() {
		//Mencetak matrix ke layar
		for (int i = 0; i < 4; i++) {
			for (int j = 0; j < 4; j++) {
				System.out.print(matrix[i][j] + " ");
			}
			System.out.println();
		}
	}

	public void move(String move) {
		//mengubah posisi angka sesuai dengan perintah yang diberikan
		Coordinate emptyPos = this.getEmptyPos();
		if (move.equals("up")) {
			int temp = this.matrix[emptyPos.getX()][emptyPos.getY()];
			this.matrix[emptyPos.getX()][emptyPos.getY()] =
this.matrix[emptyPos.getX() - 1][emptyPos.getY()];
			this.matrix[emptyPos.getX() - 1][emptyPos.getY()] = temp;
			this.step.add("up");
			this.inc_depth();
		}
		else if (move.equals("down")) {
			int temp = this.matrix[emptyPos.getX()][emptyPos.getY()];
			this.matrix[emptyPos.getX()][emptyPos.getY()] =
this.matrix[emptyPos.getX() + 1][emptyPos.getY()];
			this.matrix[emptyPos.getX() + 1][emptyPos.getY()] = temp;
			this.step.add("down");
			this.inc_depth();
		}
		else if (move.equals("left")) {
			int temp = this.matrix[emptyPos.getX()][emptyPos.getY()];
			this.matrix[emptyPos.getX()][emptyPos.getY()] =
this.matrix[emptyPos.getX()][emptyPos.getY() - 1];
			this.matrix[emptyPos.getX()][emptyPos.getY() - 1] = temp;
```

```
                        this.step.add("left");
                        this.inc_depth();
                }
                else if (move.equals("right")) {
                        int temp = this.matrix[emptyPos.getX()][emptyPos.getY()];
                        this.matrix[emptyPos.getX()][emptyPos.getY()] =
this.matrix[emptyPos.getX()][emptyPos.getY() + 1];
                        this.matrix[emptyPos.getX()][emptyPos.getY() + 1] = temp;
                        this.step.add("right");
                        this.inc_depth();
                }
        }

        public void inc_depth() {
                //menambah kedalaman simpul i
                this.depth++;
        }

        public int get_depth() {
                //mengembalikan nilai kedalaman simpul i
                return this.depth;
        }

        public int gp() {
                //ongkos mencapai simpul tujuan dari simpul i
                int count = 0;
                for (int i = 0; i < 4; i++) {
                        for (int j = 0; j < 4; j++) {
                                if ((this.matrix[i][j] != 0) && (this.matrix[i][j] !=
i*4+j+1)) {
                                        count++;
                                }
                        }
                }
                return count;
        }

        public int cost() {
                //Mengembalikan nilai cost dari suatu simpul
                return this.gp() + this.depth;
        }

        public boolean isFinish() {
                boolean isFinish = true;
                ArrayList<Integer> list = new ArrayList<Integer>();
                for (int i = 0; i < 4; i++) {
                        for (int j = 0; j < 4; j++) {
                                list.add(this.matrix[i][j]);
                        }
                }
                for (int i = 0; i < 15; i++) {
                        if (list.get(i) != i+1) {
                                isFinish = false;
                        }
                }
                return isFinish;
        }

        public String step_before() {
                //mengembalikan perintah yang dilakukan sebelumnya
                String step = "";
                if (this.step.size() == 0) {
                        step = "null";
                }
                else{
                        step = this.step.get(this.step.size() - 1);
```

```
            }
            return step;
        }

        public ArrayList<String> get_step() {
            return this.step;
        }
    }
```

b) Class Process (untuk memproses Puzzle)

```
    public class Process {
        //Class untuk memproses Matrix/Puzzle
        private Matrix mat_start;
        private Matrix mat_end;
        private double time_process;
        private int simpul;

        public Process(File file) throws FileNotFoundException {
            if (file == null) {
                this.mat_start = new Matrix();
            }
            else {
                this.mat_start = new Matrix(file);
            }
        }

        public int[] get_arr_mat_start() {
            //Mengembalikan susunan angka dalam array satu dimensi
            int[] arr = new int[16];
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    arr[i*4+j] = this.mat_start.get_element(i, j);
                }
            }
            return arr;
        }

        public int[] get_arr_mat_end() {
            //Mengembalikan susunan angka dalam array satu dimensi
            int[] arr = new int[16];
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    arr[i*4+j] = this.mat_end.get_element(i, j);
                }
            }
            return arr;
        }

        public boolean isSolvable() {
            //Mengembalikan boolean apakah matrix dapat diselesaikan atau tidak
           if ((get_kurangX()%2) != 0) {
               return false;
           }
           else {
               return true;
           }
        }

        public int get_kurangX() {
            //Mengembalikan nilai Sigma Kurang + X
            int[] kurang = this.get_kurang();
            int total = 0;
```

```
		for (int i = 0; i < 16; i++) {
			total += kurang[i];
		}
		Coordinate zero = this.mat_start.getEmptyPos();
		if ((zero.getX() == 0) || (zero.getX() == 2)) {
			if (zero.getY() == 1){
				total++;
			}
			if (zero.getY() == 3){
				total++;
			}
		}
		if ((zero.getX() == 1) || (zero.getX() == 3)){
			if (zero.getY() == 0){
				total++;
			}
			if (zero.getY() == 2){
				total++;
			}
		}
		return total;
	}

	public int[] get_kurang() {
		//Mengembalikan nilai kurang(i)
		int[] kurang = new int[16];
		int[] list = new int[16];
		int count;
		for (int i = 0; i < 4; i++) {
			for (int j = 0; j < 4; j++) {
				if (this.mat_start.get_element(i, j) == 0) {
					list[i*4+j] = 16;
				}
				else {
					list[i*4+j] = this.mat_start.get_element(i,j);

				}
			}
		}
		for (int i = 0; i < 16; i++) {
			count = 0;
			if (list[i] != 0){
				for (int j = i+1; j < 16; j++) {
					if ((list[j] < list[i]) && (list[j] != 0)){
						count++;
					}
				}
			}
			kurang[list[i]-1] = count;
		}
		return kurang;
	}

	public Matrix get_mat_start() {
		return this.mat_start;
	}

	public Matrix get_mat_end() {
		return this.mat_end;
	}

	public double get_time_process() {
		return this.time_process;
	}

	public int get_total_simpul() {
```

```
                return this.simpul;
        }

        public void process() {
                int count = 0;
                long start_time = System.nanoTime();
                if (!this.isSolvable()) {
                        //do nothing
                }
                else {
                 MyPriorityQueue pq = new MyPriorityQueue();
                 Matrix curr_mat = new Matrix(this.mat_start);
                 count++;
                 while (!curr_mat.isFinish()) {
                     Coordinate curr_zero = curr_mat.getEmptyPos();
                     if (curr_zero.getX() == 0) {
                         if (curr_zero.getY() == 0) {
                             if (curr_mat.step_before().equals("left")) {
                                 Matrix next_mat = new Matrix(curr_mat);
                                 next_mat.move("down");
                                 pq.enqueue(next_mat);
                                 count++;
                             }
                             else if (curr_mat.step_before().equals("up")) {
                                 Matrix next_mat = new Matrix(curr_mat);
                                 next_mat.move("right");
                                 pq.enqueue(next_mat);
                                 count++;
                             }
                             else{
                                 Matrix next_mat1 = new Matrix(curr_mat);
                                 next_mat1.move("down");
                                 pq.enqueue(next_mat1);
                                 count++;
                                 Matrix next_mat2 = new Matrix(curr_mat);
                                 next_mat2.move("right");
                                 pq.enqueue(next_mat2);
                                 count++;
                             }
                         }
                         else if (curr_zero.getY() == 3) {
                             if (curr_mat.step_before().equals("right")) {
                                 Matrix next_mat = new Matrix(curr_mat);
                                 next_mat.move("down");
                                 pq.enqueue(next_mat);
                                 count++;
                             }
                             else if (curr_mat.step_before().equals("up")) {
                                 Matrix next_mat = new Matrix(curr_mat);
                                 next_mat.move("left");
                                 pq.enqueue(next_mat);
                                 count++;
                             }
                             else{
                                 Matrix next_mat1 = new Matrix(curr_mat);
                                 next_mat1.move("down");
                                 pq.enqueue(next_mat1);
                                 count++;
                                 Matrix next_mat2 = new Matrix(curr_mat);
                                 next_mat2.move("left");
                                 pq.enqueue(next_mat2);
                                 count++;
                             }
                         }
                         else {
                             if (curr_mat.step_before().equals("left")) {
```

```
                            Matrix next_mat1 = new Matrix(curr_mat);
                            next_mat1.move("down");
                            pq.enqueue(next_mat1);
                            count++;
                            Matrix next_mat2 = new Matrix(curr_mat);
                            next_mat2.move("left");
                            pq.enqueue(next_mat2);
                            count++;
                        }
                        else if (curr_mat.step_before().equals("right")) {
                            Matrix next_mat1 = new Matrix(curr_mat);
                            next_mat1.move("down");
                            pq.enqueue(next_mat1);
                            count++;
                            Matrix next_mat2 = new Matrix(curr_mat);
                            next_mat2.move("right");
                            pq.enqueue(next_mat2);
                            count++;
                        }
                        else if (curr_mat.step_before().equals("up")) {
                            Matrix next_mat1 = new Matrix(curr_mat);
                            next_mat1.move("right");
                            pq.enqueue(next_mat1);
                            count++;
                            Matrix next_mat2 = new Matrix(curr_mat);
                            next_mat2.move("left");
                            pq.enqueue(next_mat2);
                            count++;
                        }
                        else{
                            Matrix next_mat1 = new Matrix(curr_mat);
                            next_mat1.move("right");
                            pq.enqueue(next_mat1);
                            count++;
                            Matrix next_mat2 = new Matrix(curr_mat);
                            next_mat2.move("down");
                            pq.enqueue(next_mat2);
                            count++;
                            Matrix next_mat3 = new Matrix(curr_mat);
                            next_mat3.move("left");
                            pq.enqueue(next_mat3);
                            count++;
                        }
                    }
                }
                else if (curr_zero.getX() == 3) {
                    if (curr_zero.getY() == 0) {
                        if (curr_mat.step_before().equals("left")) {
                            Matrix next_mat = new Matrix(curr_mat);
                            next_mat.move("up");
                            pq.enqueue(next_mat);
                            count++;
                        }
                        else if (curr_mat.step_before().equals("down")) {
                            Matrix next_mat = new Matrix(curr_mat);
                            next_mat.move("right");
                            pq.enqueue(next_mat);
                            count++;
                        }
                        else{
                            Matrix next_mat1 = new Matrix(curr_mat);
                            next_mat1.move("up");
                            pq.enqueue(next_mat1);
                            count++;
                            Matrix next_mat2 = new Matrix(curr_mat);
                            next_mat2.move("right");
```

```
                pq.enqueue(next_mat2);
                count++;
            }
        }
        else if (curr_zero.getY() == 3) {
            if (curr_mat.step_before().equals("right")) {
                Matrix next_mat = new Matrix(curr_mat);
                next_mat.move("up");
                pq.enqueue(next_mat);
                count++;
            }
            else if (curr_mat.step_before().equals("down")) {
                Matrix next_mat = new Matrix(curr_mat);
                next_mat.move("left");
                pq.enqueue(next_mat);
                count++;
            }
            else{
                Matrix next_mat1 = new Matrix(curr_mat);
                next_mat1.move("up");
                pq.enqueue(next_mat1);
                count++;
                Matrix next_mat2 = new Matrix(curr_mat);
                next_mat2.move("left");
                pq.enqueue(next_mat2);
                count++;
            }
        }
        else {
            if (curr_mat.step_before().equals("left")) {
                Matrix next_mat1 = new Matrix(curr_mat);
                next_mat1.move("up");
                pq.enqueue(next_mat1);
                count++;
                Matrix next_mat2 = new Matrix(curr_mat);
                next_mat2.move("left");
                pq.enqueue(next_mat2);
                count++;
            }
            else if (curr_mat.step_before().equals("right")) {
                Matrix next_mat1 = new Matrix(curr_mat);
                next_mat1.move("up");
                pq.enqueue(next_mat1);
                count++;
                Matrix next_mat2 = new Matrix(curr_mat);
                next_mat2.move("right");
                pq.enqueue(next_mat2);
                count++;
            }
            else if (curr_mat.step_before().equals("down")) {
                Matrix next_mat1 = new Matrix(curr_mat);
                next_mat1.move("right");
                pq.enqueue(next_mat1);
                count++;
                Matrix next_mat2 = new Matrix(curr_mat);
                next_mat2.move("left");
                pq.enqueue(next_mat2);
                count++;
            }
            else{
                Matrix next_mat1 = new Matrix(curr_mat);
                next_mat1.move("right");
                pq.enqueue(next_mat1);
                count++;
                Matrix next_mat2 = new Matrix(curr_mat);
                next_mat2.move("up");
```

```
                    pq.enqueue(next_mat2);
                    count++;
                    Matrix next_mat3 = new Matrix(curr_mat);
                    next_mat3.move("left");
                    pq.enqueue(next_mat3);
                    count++;
                }
            }
        }
        else{
            //di tengah
            if (curr_zero.getY() == 0) {
                if (curr_mat.step_before().equals("left")) {
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("right");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("down");
                    pq.enqueue(next_mat2);
                    count++;
                }
                else if (curr_mat.step_before().equals("down")) {
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("down");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("right");
                    pq.enqueue(next_mat2);
                    count++;
                }
                else if (curr_mat.step_before().equals("up")) {
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("up");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("right");
                    pq.enqueue(next_mat2);
                    count++;
                }
                else{
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("up");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("right");
                    pq.enqueue(next_mat2);
                    count++;
                    Matrix next_mat3 = new Matrix(curr_mat);
                    next_mat3.move("down");
                    pq.enqueue(next_mat3);
                    count++;
                }
            }
            else if (curr_zero.getY() == 3) {
                if (curr_mat.step_before().equals("right")) {
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("up");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("down");
                    pq.enqueue(next_mat2);
```

```
                    count++;
                }
                else if (curr_mat.step_before().equals("up")) {
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("up");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("left");
                    pq.enqueue(next_mat2);
                    count++;
                }
                else if (curr_mat.step_before().equals("down")) {
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("down");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("left");
                    pq.enqueue(next_mat2);
                    count++;
                }
                else{
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("down");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("left");
                    pq.enqueue(next_mat2);
                    count++;
                    Matrix next_mat3 = new Matrix(curr_mat);
                    next_mat3.move("up");
                    pq.enqueue(next_mat3);
                    count++;
                }
            }
            else {
                if (curr_mat.step_before().equals("left")) {
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("left");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("up");
                    pq.enqueue(next_mat2);
                    count++;
                    Matrix next_mat3 = new Matrix(curr_mat);
                    next_mat3.move("down");
                    pq.enqueue(next_mat3);
                    count++;
                }
                else if (curr_mat.step_before().equals("right")) {
                    Matrix next_mat1 = new Matrix(curr_mat);
                    next_mat1.move("right");
                    pq.enqueue(next_mat1);
                    count++;
                    Matrix next_mat2 = new Matrix(curr_mat);
                    next_mat2.move("up");
                    pq.enqueue(next_mat2);
                    count++;
                    Matrix next_mat3 = new Matrix(curr_mat);
                    next_mat3.move("down");
                    pq.enqueue(next_mat3);
                    count++;
                }
```

```
                        else if (curr_mat.step_before().equals("up")) {
                            Matrix next_mat1 = new Matrix(curr_mat);
                            next_mat1.move("up");
                            pq.enqueue(next_mat1);
                            count++;
                            Matrix next_mat2 = new Matrix(curr_mat);
                            next_mat2.move("left");
                            pq.enqueue(next_mat2);
                            count++;
                            Matrix next_mat3 = new Matrix(curr_mat);
                            next_mat3.move("right");
                            pq.enqueue(next_mat3);
                            count++;
                        }
                        else if (curr_mat.step_before().equals("down")) {
                            Matrix next_mat1 = new Matrix(curr_mat);
                            next_mat1.move("down");
                            pq.enqueue(next_mat1);
                            count++;
                            Matrix next_mat2 = new Matrix(curr_mat);
                            next_mat2.move("left");
                            pq.enqueue(next_mat2);
                            count++;
                            Matrix next_mat3 = new Matrix(curr_mat);
                            next_mat3.move("right");
                            pq.enqueue(next_mat3);
                            count++;
                        }
                        else{
                            Matrix next_mat1 = new Matrix(curr_mat);
                            next_mat1.move("up");
                            pq.enqueue(next_mat1);
                            count++;
                            Matrix next_mat2 = new Matrix(curr_mat);
                            next_mat2.move("left");
                            pq.enqueue(next_mat2);
                            count++;
                            Matrix next_mat3 = new Matrix(curr_mat);
                            next_mat3.move("right");
                            pq.enqueue(next_mat3);
                            count++;
                            Matrix next_mat4 = new Matrix(curr_mat);
                            next_mat4.move("down");
                            pq.enqueue(next_mat4);
                            count++;
                        }
                    }
                }
                curr_mat = pq.dequeue();
            }
            //get curr_mat as result
            this.mat_end = new Matrix(curr_mat);
           }
        long end_time = System.nanoTime();
           //in milisecond
           this.time_process = (end_time-start_time) / 1000000;
           this.simpul = count;
    }
}
```

c) Class GUI

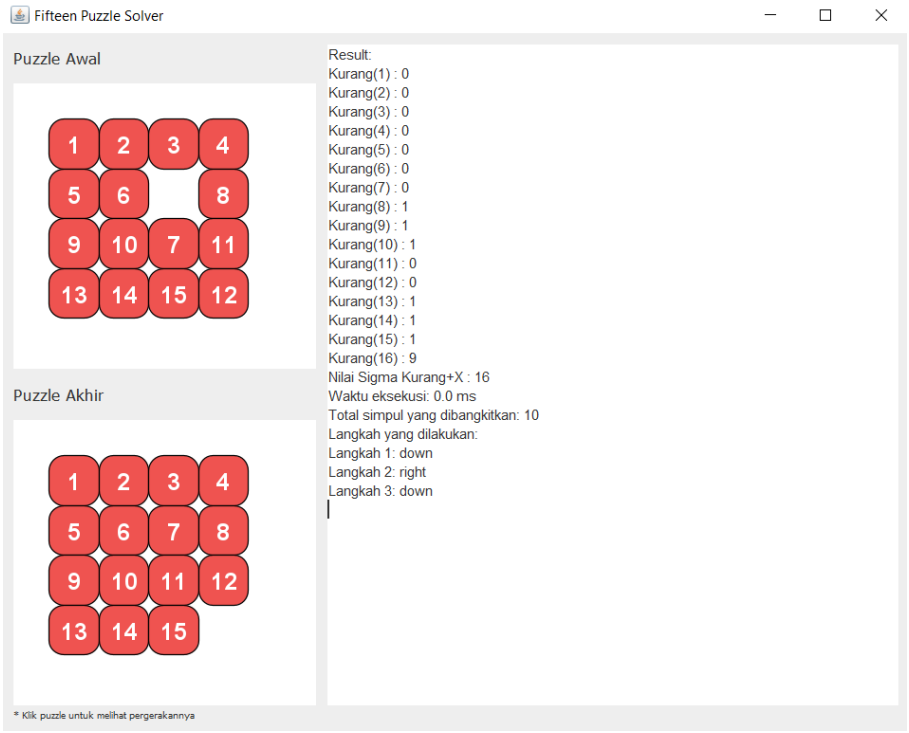
```
public class Start extends JFrame {
    //Start GUI
```

```
	private JPanel contentPane;
	private JTextField textField;
	private File file;
	private Process puzzle;

	public Start() {
		setTitle("Fifteen Puzzle Solver");
		setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		setBounds(100, 100, 700, 500);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(null);

		JLabel lblNewLabel = new JLabel("Fifteen Puzzle Solver");
		lblNewLabel.setFont(new Font("Tekton Pro Cond", Font.BOLD, 40));
		lblNewLabel.setBounds(202, 92, 282, 60);
		contentPane.add(lblNewLabel);

		JRadioButton rdbtnNewRadioButton = new JRadioButton("Random
Sample");
		rdbtnNewRadioButton.setFont(new Font("Tahoma", Font.PLAIN, 16));
		rdbtnNewRadioButton.setBounds(272, 177, 141, 46);
		contentPane.add(rdbtnNewRadioButton);

		JRadioButton rdbtnNewRadioButton_1 = new JRadioButton("Choose
File");
		rdbtnNewRadioButton_1.setFont(new Font("Tahoma", Font.PLAIN,
16));
		rdbtnNewRadioButton_1.setBounds(272, 234, 133, 39);
		contentPane.add(rdbtnNewRadioButton_1);

		ButtonGroup buttonGroup = new ButtonGroup();
		buttonGroup.add(rdbtnNewRadioButton);
		buttonGroup.add(rdbtnNewRadioButton_1);

		textField = new JTextField();
		textField.setBounds(270, 274, 146, 24);
		contentPane.add(textField);
		textField.setColumns(10);

		JButton btnNewButton = new JButton("Choose FIle");
		btnNewButton.setFont(new Font("Tahoma", Font.PLAIN, 12));
		btnNewButton.setBounds(294, 308, 97, 23);
		btnNewButton.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if (rdbtnNewRadioButton_1.isSelected()) {
					if(e.getSource()==btnNewButton) {
						JFileChooser fileChooser = new
JFileChooser();

						fileChooser.setCurrentDirectory(new
File(".")); //sets current directory
						int response =
fileChooser.showOpenDialog(null); //select file to open
						if(response ==
JFileChooser.APPROVE_OPTION) {
							file = new
File(fileChooser.getSelectedFile().getAbsolutePath());
							try {
								puzzle = new
Process(file);
							} catch
(FileNotFoundException e1) {

    e1.printStackTrace();
```

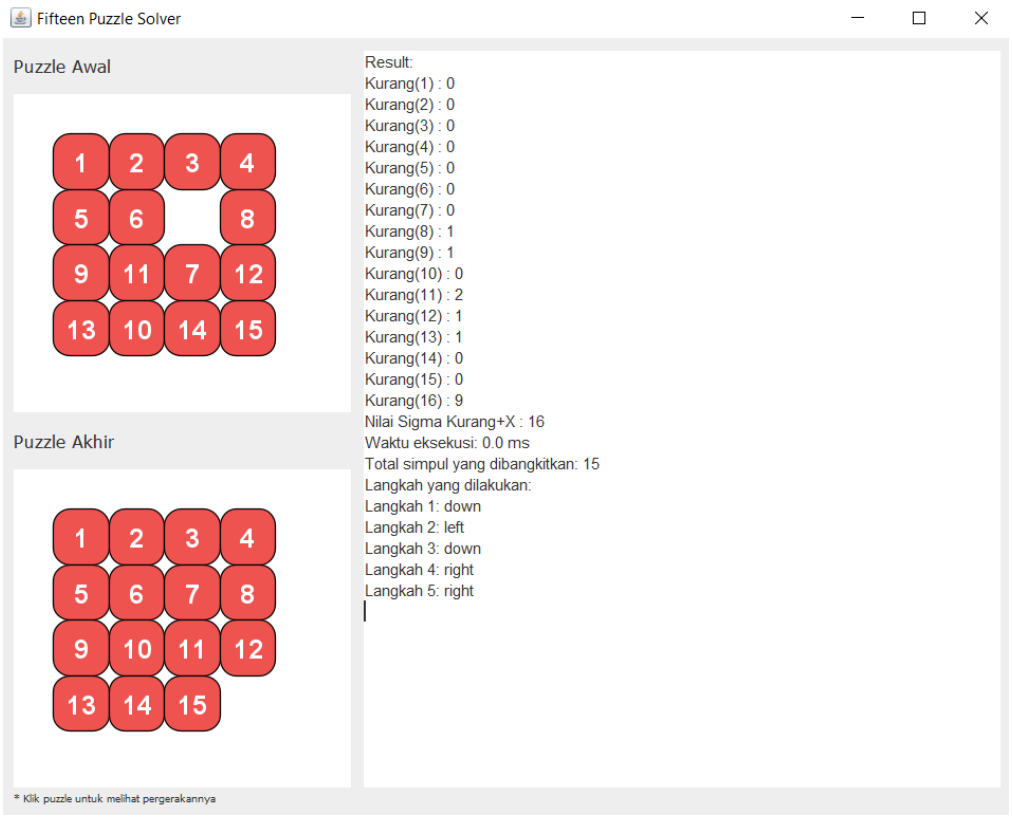
```
                                                        }

        textField.setText(file.toString());
                                                    }
                                                }
                                            }
                                        }
                                });

                                try {
                                        puzzle = new Process(null);
                                } catch (FileNotFoundException e1) {
                                        e1.printStackTrace();
                                }
                                contentPane.add(btnNewButton);

                                JButton btnNewButton_1 = new JButton("Solve!!!");
                                btnNewButton_1.setFont(new Font("Tahoma", Font.PLAIN, 14));
                                btnNewButton_1.setBounds(283, 408, 120, 33);
                                contentPane.add(btnNewButton_1);
                                btnNewButton_1.addActionListener(new ActionListener() {
                                        public void actionPerformed(ActionEvent e) {
                                                if (rdbtnNewRadioButton_1.isSelected() ||
    rdbtnNewRadioButton.isSelected()) {
                                                if(e.getSource()==btnNewButton_1) {
                                                        Result res = new Result(puzzle);
                                                        res.setVisible(true);
                                                        dispose();
                                                }
                                        }
                                    }
                                });
        }
    }
```

3. Screen Input-Output Program

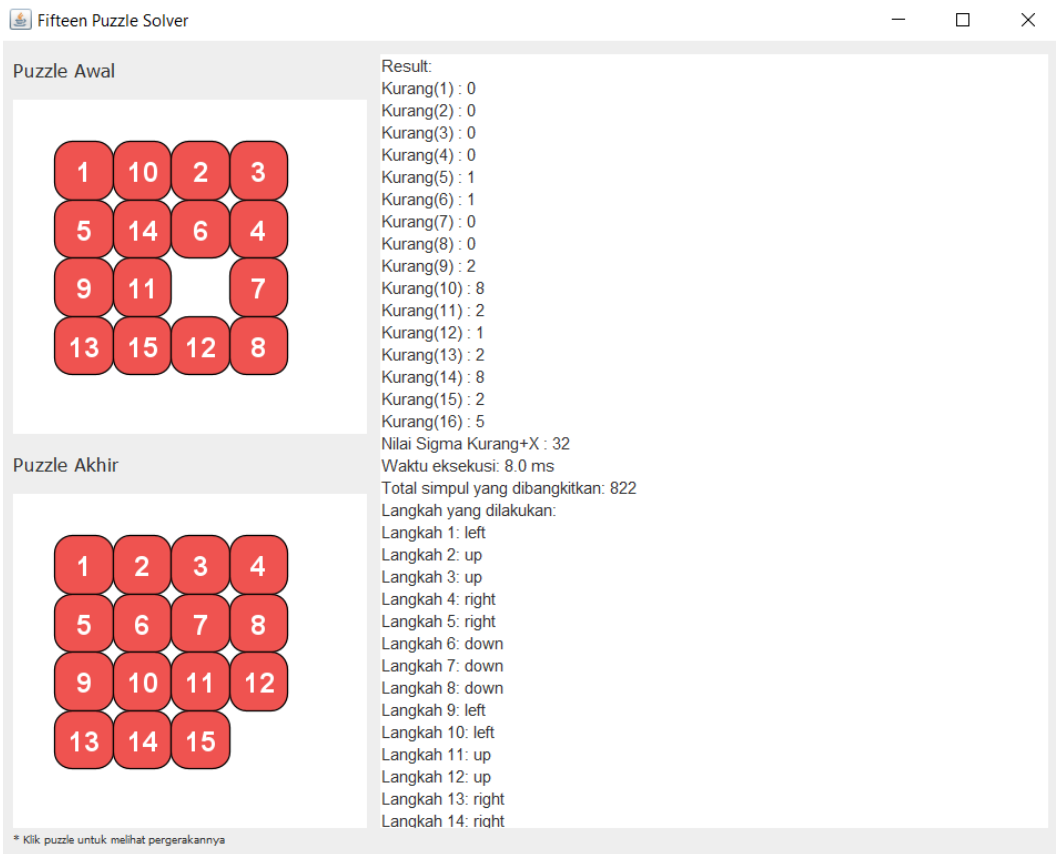
a. test1.txt (bisa diselesaikan)



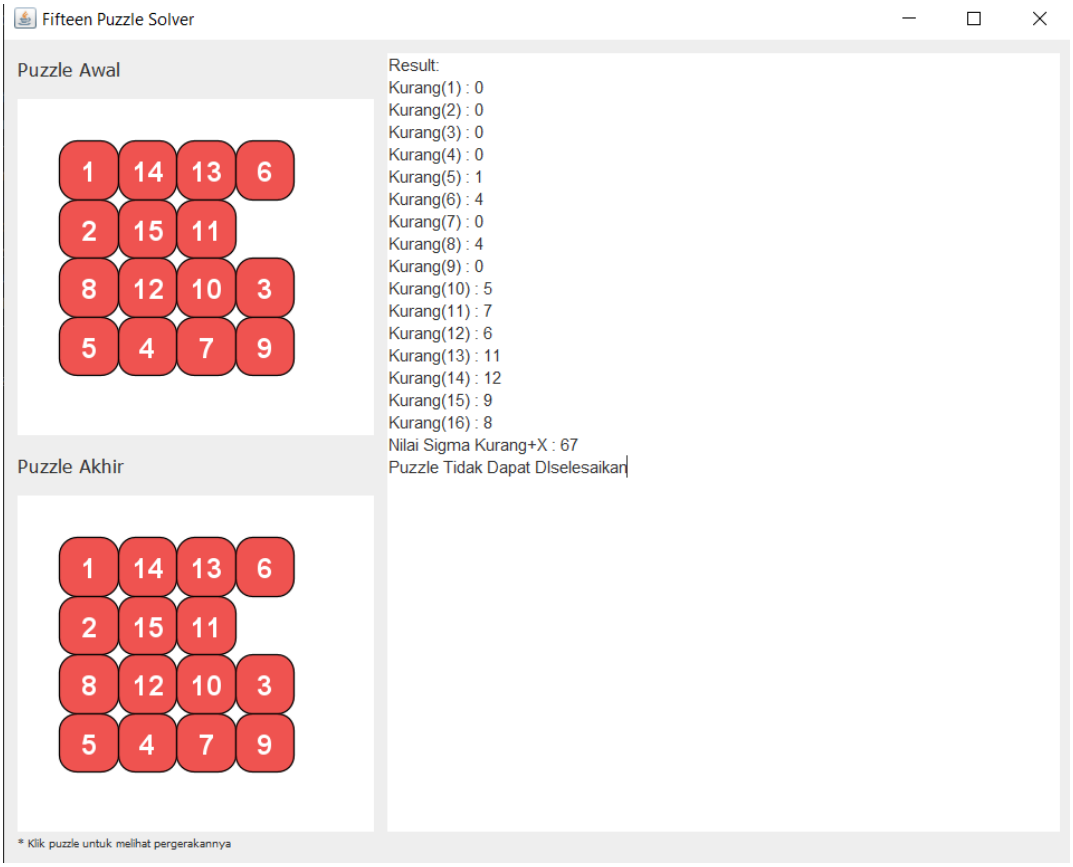
b. test2.txt (bisa diselesaikan)



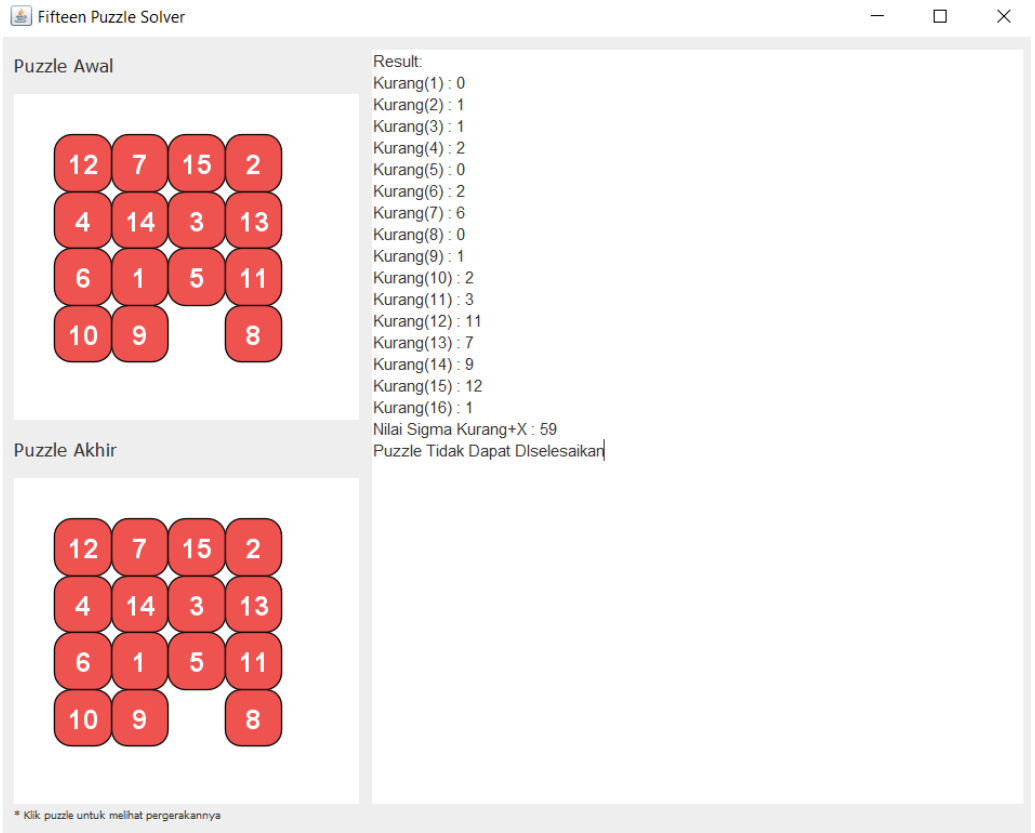
c. test3.txt (bisa diselesaikan)



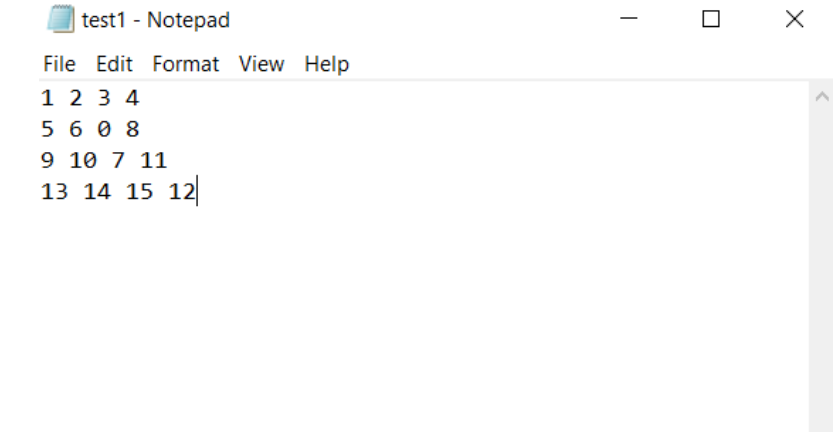
d. test4.txt (tidak bisa diselesaikan)

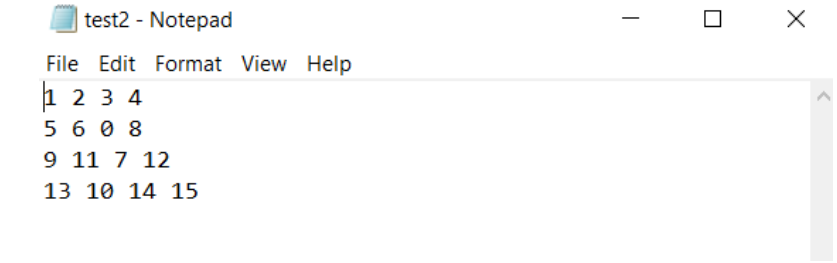


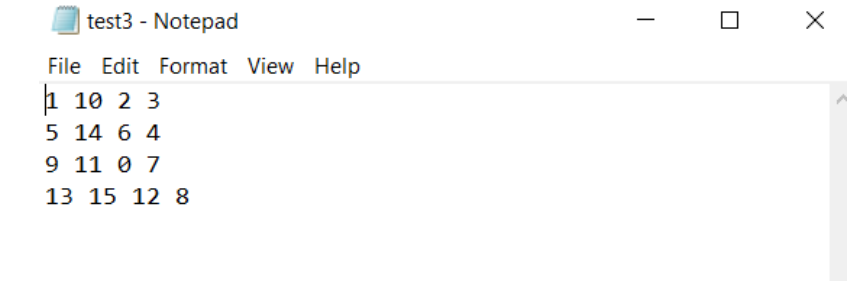
e. test5.txt (tidak bisa diselesaikan)

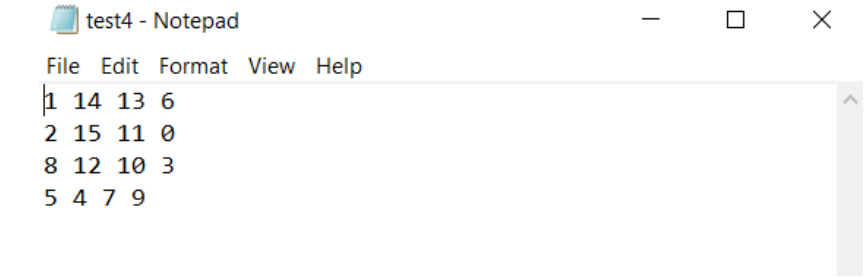


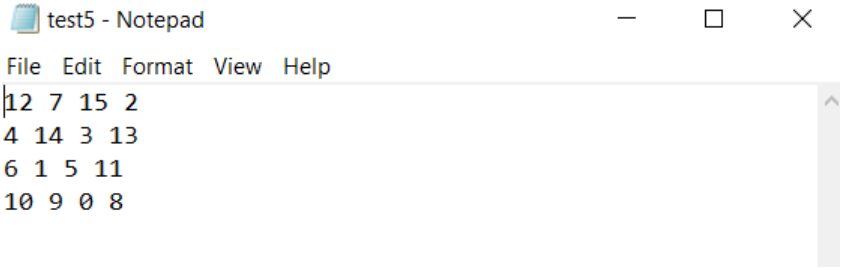
4. Berkas Text Input Program











5. **Drive Program**

   https://github.com/haidarihza/Tucil3_13520150

Checklist Poin:

Poin	Ya	Tidak
1. Program berhasil dikompilasi	✓	
2. Program berhasil *running*	✓	
3. Program dapat menerima input dan menuliskan output	✓	
4. Luaran sudah benar untuk semua data uji	✓	
5. Bonus dibuat	✓	