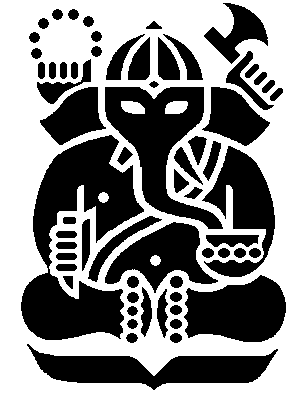
**Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch and Bound***

LAPORAN TUGAS KECIL

Diajukan Untuk Memenuhi Tugas Kecil IF2211 Strategi Algoritma

Semester II 2021/2022



Disusun oleh:

Jason Kanggara

13520080

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2022**

**A. Algoritma *Branch and Bound***

Algoritma *Branch and Bound* adalah algoritma yang digunakan untuk persoalan optimasi, yaitu meminimalkan atau memaksimalkan suatu fungsi objektif yang tidak melanggar batasa persoalan. algoritma *Branch and Bound* dapat dianggap sebagai gabungan antara BFS (*Breadth First Search*) dengan *Least Cost Search*. Jadi, pada algoritma *Branch and Bound*, setiap simpul diberi nilai *cost* dan simpul berikutnya yang akan di-*expand* diurut berdasarkan simpul yang memiliki *cost* paling kecil (*pada kasus minimasi*). Pada tugas kecil kali ini, algoritma *Branch and Bound* akan digunakan untuk menyelesaikan persoalan 15-Puzzle.

Pada persoalan 15-Puzzle ini, langkah – langkah algoritma yang digunakan adalah sebagai berikut:

1. Tentukan apakah *puzzle* dapat mencapai solusi/*goal* dengan menggunakan metode sum(KURANG(i)) + x.
2. Jika sum(KURANG(i)) + x bernilai ganjil, **Stop**. Jika bernilai genap, lanjutkan ke langkah 3.
3. Jika simpul akar adalah solusi, maka solusi telah ditemukan dan program langsung **Stop**. Jika tidak, maka lanjut ke langkah 4.
4. Ekspansi simpul akar, hitung *cost* dari setiap simpul hasil ekspansi, lalu masukkan masing – masing simpul ke antrian berupa Priority Queue dengan memprioritaskan simpul yang memiliki *cost* paling kecil.
5. Ambil simpul dengan *cost* terkecil pada antrian lalu hapus simpul tersebut dari antrian. Jika simpul yang diambil adalah solusi, maka **Stop**. Jika tidak, kembail ke langkah 4 dengan mengganti simpul akar dengan simpul yang diambil dari antrian.

**B. Source Code**

|  |
| --- |
| **PuzzleSolverGUI.java** |
| import javax.swing.\*;  import java.awt.\*;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  import lib.\*;  public class PuzzleSolverGUI extends JFrame {      private static int size = 0;      private static boolean solvable;      private int[][] matrix;      private int[][] solution;      private Solver puzzle;      private JPanel mainPanel;      private JLabel titleLabel;      private JTextField inputField;      private JButton VIEWPUZZLEButton;      private JButton SOLVEButton;      private JPanel outputPanel;      private JLabel puzzleStatus;      private JLabel puzzleView;      private JLabel moveLabel;      private JLabel verticesRaised;      private final Timer timer;      private int iter;      private String executionTime;      public PuzzleSolverGUI(String title) {          super(title);          this.titleLabel.setFont(new Font("SansSerif", Font.PLAIN, 28));          this.puzzleView.setFont(new Font("SansSerif", Font.PLAIN, 32));          this.puzzleStatus.setFont(new Font("SansSerif", Font.PLAIN, 16));          this.moveLabel.setFont(new Font("SansSerif", Font.BOLD, 16));          this.verticesRaised.setFont(new Font("SansSerif", Font.BOLD, 16));          this.inputField.setFont(new Font("SansSerif", Font.PLAIN, 18));          this.SOLVEButton.setEnabled(false);          this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);          this.setContentPane(mainPanel);          this.setSize(420, 480);          this.setResizable(false);          // Action saat menekan tombol VIEWPUZZLE          VIEWPUZZLEButton.addActionListener(new ActionListener() {              @Override              public void actionPerformed(ActionEvent e) {                  // inisialisasi String, matrix, dan matrix solusi/goal                  String initialPuzzle;                  matrix = null;                  solution = null;                  Solver.antrian.clear();                  Solver.path.clear();                  Solver.solutions.clear();                  // Mengambil text dari input field sebagai fileName                  String fileName = inputField.getText();                  // Membaca matrix dan solusi                  try {                      matrix = readMatrix("../test/" + fileName);                      solution = readMatrix("solution.txt");                  } catch (IOException err) {                      System.out.println("Error: " + err.getMessage());                  }                  // inisialisasi Solver                  puzzle = new Solver(matrix, solution, size);                  moveLabel.setText("");                  verticesRaised.setText("");                  // Memeriksa jika puzzle dapat mencapai goal atau tidak                  if (puzzle.isGoalReachable()) {                      puzzleStatus.setText("Puzzle memiliki solusi");                      solvable = true;                  } else {                      puzzleStatus.setText("Puzzle tidak memiliki solusi");                      solvable = false;                  }                  // Jika solvable, maka SOLVEButton dapat ditekan dan puzzle awal ditunjukkan                  if (solvable) {                      initialPuzzle = printMatrixToString(matrix);                      puzzleView.setText(initialPuzzle);                      SOLVEButton.setEnabled(true);                  }              }          });          // Action saat menekan SOLVEButton          SOLVEButton.addActionListener(new ActionListener() {              @Override              public void actionPerformed(ActionEvent e) {                  puzzleStatus.setText("Calculating...");                  // jalankan method solve puzzle                  puzzle.Solve();                  executionTime = Long.toString(Solver.execTime);                  if (Solver.path.size() == 0) {                      verticesRaised.setText("Vertices Raised: " + 0);                      puzzleStatus.setText("<html>Steps taken: " + 0 + " steps<br/>Execution Time: " + executionTime + "ms</html>");                      puzzleView.setText(printMatrixToString(matrix));                      moveLabel.setText("none");                  } else {                      iter = Solver.path.size() - 1;                      timer.start();                  }                  SOLVEButton.setEnabled(false);              }          });          timer = new Timer(800, new ActionListener() {              @Override              public void actionPerformed(ActionEvent e) {                  if (iter == 0) {                      timer.stop();                      verticesRaised.setText("Vertices Raised: " + Solver.solutions.size());                      puzzleStatus.setText("<html>Steps taken: " + Solver.path.size() + " steps<br/>Execution Time: " + executionTime + "ms</html>");                  }                  // menampilkan puzzle dengan delay                  puzzleView.setText(printMatrixToString(Solver.path.get(iter).matrix));                  moveLabel.setText(Solver.path.get(iter).move);                  iter--;              }          });      }      // Fungsi untuk membaca matrix dari suatu file      public static int[][] readMatrix(String filename) throws IOException {          int[][] matrix = null;          BufferedReader buffer = new BufferedReader(new FileReader(filename));          String line;          int row = 0;          while ((line = buffer.readLine()) != null) {              String[] vals = line.trim().split("\\s+");              // Instansiasi matriks              if (matrix == null) {                  size = vals.length;                  matrix = new int[size][size];              }              for (int col = 0; col < size; col++) {                  matrix[row][col] = Integer.parseInt(vals[col]);              }              row++;          }          buffer.close();          return matrix;      }      // Fungsi untuk mengubah suatu matrix menjadi string yang dapat ditampilkan di GUI      public String printMatrixToString(int[][] matrix) {          StringBuilder sb = new StringBuilder();          sb.append("<html>");          for (int i = 0; i < size; i++) {              for (int j = 0; j < size; j++) {                  if (matrix[i][j] == 16) {                      sb.append("| \_ |");                  } else if (matrix[i][j] < 10) {                      sb.append("| " + matrix[i][j] + " |");                  } else {                      sb.append("|" + matrix[i][j] + "|");                  }              }              sb.append("<br />");          }          sb.append("</html>");          return sb.toString();      }      public static void main(String[] args) {          JFrame frame = new PuzzleSolverGUI("15 Puzzle Solver");          frame.setVisible(true);      }  } |

|  |
| --- |
| **Solver.java** |
| package lib;  import java.util.ArrayList;  import java.util.List;  import java.util.PriorityQueue;  // Class Solver sebagai class utama untuk mencari solusi  public class Solver {    // variable matrix untuk menyimpan matrix yang dites    private int[][] matrix;    // variable solution untuk menyimpan matrix keadaan terakhir (goal dari puzzle)    private int[][] solution;    // variable size sebagai ukuran matrix (4x4)    private int size;    // antrian sebagai prioqueue berdasarkan cost dari Node    public static PriorityQueue<Node> antrian = new PriorityQueue<Node>(new NodeComparator());    // List of solutions dari hasil pop prioqueue    public static List<Node> solutions = new ArrayList<Node>();    // variable execution time    public static long execTime;    // List of path berupa jalur dari matrix awal ke goal    public static List<Node> path = new ArrayList<Node>();    // Constructor dari Solver    public Solver(int[][] matrix, int[][] solution, int size) {      this.matrix = matrix;      this.solution = solution;      this.size = size;    }    // Get elemen matrix    public int Elmt(int i, int j) {      return this.matrix[i][j];    }    // Get elemen solusi    public int Sol(int i, int j) {      return this.solution[i][j];    }    // get index row dari elemen empty    public int getEmptyRow() {      for (int i = 0; i < size; i++) {        for (int j = 0; j < size; j++) {          if (Elmt(i, j) == 16) {            return i;          }        }      }      return -999;    }    // get index column dari elemen empty    public int getEmptyCol() {      for (int i = 0; i < size; i++) {        for (int j = 0; j < size; j++) {          if (Elmt(i, j) == 16) {            return j;          }        }      }      return -999;    }    // print matrix ke console/terminal    public void printInfo(int[][] matrix) {      for (int i = 0; i < size; i++) {        for (int j = 0; j < size; j++) {          if (matrix[i][j] == 16) {            System.out.print("   ");          } else if (matrix[i][j] < 10) {            System.out.print(" " + matrix[i][j] + " ");          } else {            System.out.print(matrix[i][j] + " ");          }        }        System.out.println();      }    }    // menentukan value dari x    // jika sel kosong di posisi yang diarsir, maka x = 1;    // else x = 0;    public int valueOfX() {      for (int i = 0; i < size; i++) {        for (int j = 0; j < size; j++) {          if (Elmt(i, j) == 16) {            if ((i+1) % 2 == 1) {              if ((j+1) % 2 == 0) {                return 1;              } else {                return 0;              }            } else if ((i+1) % 2 == 0) {              if ((j+1) % 2 == 1) {                return 1;              } else {                return 0;              }            }          }        }      }      return -999;    }    // mengubah matrix menjadi array 1 dimensi    public int[] convertToOneD(int[][] matrix) {      List<Integer> list = new ArrayList<Integer>();      for (int i = 0; i < size; i++) {        for (int j = 0; j < size; j++) {          list.add(Elmt(i, j));        }      }      int[] oneDArray = new int[list.size()];      for (int i = 0; i < oneDArray.length; i++) {        oneDArray[i] = list.get(i);      }      return oneDArray;    }    // kalkulasi value dari kurang,    // yaitu banyaknya ubin yang valuenya kurang dari ubin acuan    public int KURANG(int x, int position) {      int count = 0;      int[] list = convertToOneD(matrix);      for (int i = position; i < list.length; i++) {        int temp = list[i];        if (temp < x) {          count++;        }      }      return count;    }    // mencari hasil dari sumOf(KURANG)    public int valueOfKurang() {      int sum = 0;      int count = 0;      int position = 0;      for (int row = 0; row < size; row++) {        for (int col = 0; col < size; col++) {          int temp = Elmt(row, col);          count = KURANG(temp, position);          position += 1;          sum += count;        }      }      return sum;    }    // Menentukan apakah puzzle dapat mencapai goal atau tidak    // - jika KURANG + x bernilai genap maka reachable    // - else tidak reachable    public boolean isGoalReachable() {      // isGoalReachable menggunakan rumus KURANG(i) + X;        // Pencarian nilai x      int x = valueOfX();      // Pencarian total dari KURANG(i)      int kurang = valueOfKurang();      return (kurang + x) % 2 == 0;    }    // get kemungkinan moves yang dapat dilakukan oleh sel kosong berdasarkan posisinya    public String[] getPossibleMoves() {      List<String> list = new ArrayList<String>();      String up = "up";      String right = "right";      String down = "down";      String left = "left";      int row = getEmptyRow();      int col = getEmptyCol();      if (row == 0 && col == 0) {        list.add(right);        list.add(down);      } else if (row == 0 && col == size - 1) {        list.add(down);        list.add(left);      } else if (row == size - 1 && col == 0) {        list.add(up);        list.add(right);      } else if (row == size - 1 && col == size - 1) {        list.add(up);        list.add(left);      } else if (row == 0 && col > 0 && col < size) {        list.add(right);        list.add(down);        list.add(left);      } else if (col == 0 && row > 0 && row < size) {        list.add(up);        list.add(right);        list.add(down);      } else if (row == size - 1 && col > 0 && col < size) {        list.add(up);        list.add(right);        list.add(left);      } else if (col == size - 1 && row > 0 && row < size) {        list.add(up);        list.add(down);        list.add(left);      } else {        list.add(up);        list.add(right);        list.add(down);        list.add(left);      }      String[] arrayOfMoves = new String[list.size()];      for (int i = 0; i < arrayOfMoves.length; i++) {        arrayOfMoves[i] = list.get(i);      }      return arrayOfMoves;    }    // mengassign suatu matrix dengan matrix lain    public int[][] copyMatrix(int[][] inputMat) {      int[][] newMatrix = new int[size][size];      for (int i = 0; i < size; i++) {        for (int j = 0; j < size; j++) {          newMatrix[i][j] = inputMat[i][j];        }      }      return newMatrix;    }    // untuk melakukan swapping antara sel kosong dan sel sekitarnya berdasarkan move    public void swapEmpty(int rowSrc, int colSrc, int rowDest, int colDest) {      int temp = this.matrix[rowSrc][colSrc];      this.matrix[rowSrc][colSrc] = this.matrix[rowDest][colDest];      this.matrix[rowDest][colDest] = temp;    }    // command move untuk menggerakkan sel kosong    public void move(String command) {      int rowEmpty = getEmptyRow();      int colEmpty = getEmptyCol();      if (command.equals("up")) {        swapEmpty(rowEmpty, colEmpty, rowEmpty - 1, colEmpty);      } else if (command.equals("down")) {        swapEmpty(rowEmpty, colEmpty, rowEmpty + 1, colEmpty);      } else if (command.equals("right")) {        swapEmpty(rowEmpty, colEmpty, rowEmpty, colEmpty + 1);      } else if (command.equals("left")) {        swapEmpty(rowEmpty, colEmpty, rowEmpty, colEmpty - 1);      }    }    // perhitungan cost suatu puzzle    public int countCost(int[][] inputMat, int depth) {      int count = 0;      for (int i = 0; i < size; i++) {        for (int j = 0; j < size; j++) {          if (inputMat[i][j] != 16) {            if (inputMat[i][j] != this.solution[i][j]) {              count++;            }          }        }      }      return count + depth;    }    // melakukan pengecekan puzzle dengan solusi/goal    // - jika ada elemen yang tidak sama, maka notSolution mengembalikan true    // - jika semua elemen sama, maka matrix tersebut adalah solution    public boolean notSolution(int[][] matrix) {      for (int i = 0; i < size; i++) {        for (int j = 0; j < size; j++) {          if (matrix[i][j] != solution[i][j]) {            return true;          }        }      }      return false;    }    // memeriksa apakah kedua matrix yang dibandingkan sama    // jika sama maka true, else false    public boolean isSame(int[][] matrix, int[][] initialMatrix) {      for (int i = 0; i < size; i++) {        for (int j = 0; j < size; j++) {          if (matrix[i][j] != initialMatrix[i][j]) {            return false;          }        }      }      return true;    }    // mengembalikan true atau false berdasarkan masukkan command move    // agar puzzle yang akan diekspansi tidak melakukan move yang akan mengembalikan dia    // ke posisi semula    public boolean returnTheSame(String first, String second) {      if (first == "left" && second == "right") {        return true;      } else if (first == "right" && second == "left") {        return true;      } else if (first == "down" && second == "up") {        return true;      } else if (first == "up" && second == "down") {        return true;      }      return false;    }    // Main Solving Method    public void Solve() {      int level = 1;      int cost;      String firstMove = "none";      long startTime = System.currentTimeMillis();      // Jika puzzle sudah solusi, maka langsung return;      if (!notSolution(this.matrix)) {        long stopTime = System.currentTimeMillis();        execTime = stopTime - startTime;        return;      }      // while matrix tidak sama dengan solusi/goal, loop      while(notSolution(this.matrix)) {        int[][] initialMatrix = copyMatrix(this.matrix);        String[] possibleMoves = getPossibleMoves();        for (int i = 0; i < possibleMoves.length; i++) {          move(possibleMoves[i]);          if (!returnTheSame(firstMove, possibleMoves[i])) {            cost = countCost(this.matrix, level);            // Memasukkan Node puzzle ke antrian            antrian.add(new Node(this.matrix, initialMatrix, level, cost, possibleMoves[i]));          }          this.matrix = copyMatrix(initialMatrix);        }        // Mengambil Node puzzle pertama pada prioqueue        Node nextMove = antrian.poll();        firstMove = nextMove.move;        if (nextMove.level < level || nextMove.level > level) {          level = nextMove.level;        }        level += 1;        // Memasukkan Node puzzle yang menjadi move berikutnya ke list of solusi        solutions.add(nextMove);        this.matrix = copyMatrix(nextMove.matrix);        // jika puzzle nextMove berupa solusi, maka dimasukkan ke path        if (!notSolution(nextMove.matrix)) {          path.add(nextMove);        }      }      long stopTime = System.currentTimeMillis();      // pencarian jalur dengan menghubungkan puzzle dengan parent dari elemen terakhir path      for (int i = solutions.size() - 1; i >= 0; i--) {        if (isSame(solutions.get(i).matrix, path.get(path.size() - 1).parent)) {          path.add(solutions.get(i));        }      }      // perhitungan ekesekusi waktu      execTime = stopTime - startTime;    }  } |

|  |
| --- |
| **Node.java** |
| package lib;  // Class Node sebagai identitas simpul setiap puzzle  public class Node {    // matrix untuk menyimpan bentuk puzzle    public int[][] matrix;    // parent untuk menyimpan puzzle induknya    public int[][] parent;    // level untuk menyimpan tingkatan/kedalaman dari simpul puzzle    public int level;    // cost untuk menyimpan cost pada simpul puzzle    public int cost;    // move untuk menyimpan move apa yang digunakan untuk mencapai simpul this    public String move;    // Constructor dari Node dengan parameter    public Node(int[][] matrix, int[][] parent, int level, int cost, String move) {      this.matrix = matrix;      this.parent = parent;      this.level = level;      this.cost = cost;      this.move = move;    }  } |

|  |
| --- |
| **NodeComparator.java** |
| package lib;  import java.util.Comparator;  // Class NodeComparator untuk mengOverride comparator pada PriorityQueue untuk  // menyesuaikan dengan objek buatan bernama Node  // Comparasi dilakukan dengan membandingkan cost:  // - cost kecil memiliki prioritas tertinggi dan dimasukkan menjadi elemen pertama prioqueue  public class NodeComparator implements Comparator<Node> {    @Override    public int compare(Node n1, Node n2) {      if (n1.cost < n2.cost) {        return -1;      } else if (n1.cost > n2.cost) {        return 1;      }      return 0;    }  } |

**C. Hasil Program**

|  |  |  |
| --- | --- | --- |
| **No** | **Initial Puzzle** | **Puzzle Result** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

**D. Link Source Code**

**E. Tabel Cek List**

|  |  |  |
| --- | --- | --- |
| 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 uji | √ |  |
| 1. Bonus dibuat | √ |  |