**LAPORAN TUGAS KECIL-3 STRATEGI ALGORITMA**

**Program penyelesaian 15-puzzle dengan Algoritma Branch and Bound**



**Disusun Oleh:**

Muhammad Gerald Akbar Giffera 13520143

**Institut Teknologi Bandung**

**2022**

1. **Algoritma *Branch and Bound***

A picture containing text, electronics, calculator

Description automatically generated

Algoritma *branch and bound,* adalah algoritma digunakan untuk menyelesaikan persoalan optimalisasi (minimalisasi/maksimalisasi). Algoritma ini, mirip dengan algoritma *BFS,* karena sama-sama memanfaatkan struktur data *queue.* Sama seperti halnya pada BFS setiap simpul yang dibangkitkan yang akan dilakukan analisis lebih lanjut akan disimpan ke dalam queue, akan tetapi terdapat sedikit perbedaan pada *branch and bound,* yaitu pada proses mengkases queue, karena pada *branch and bound* setiap simpul memiliki nilai/*cost* tertentu yang akan menyatakan urutan aksesnya, sehingga bisa dibilang bahwa lebih tepatnya algoritma branch and bound memanfaatkan struktur data *priority queue.*

Secara garis besar, terdapat langkah-langkah yang dilakukan dalam algoritma *branch and bound*, yaitu sebagai berikut:

1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi (goal node), maka solusi telah ditemukan. Jika hanya satu solusi yang diinginkan, maka stop.
2. Jika Q kosong, Stop.
3. Jika Q tidak kosong, pilih dari antrian Q simpul i yang mempunyai nilai ‘cost’ ĉ(i) paling kecil. Jika terdapat beberapa simpul i yang memenuhi, pilih satu secara sembarang.
4. Jika simpul i adalah simpul solusi, berarti solusi sudah ditemukan. Jika satu solusi yang diinginkan, maka stop. Pada persoalan optimasi dengan pendekatan least cost search, periksa cost semua simpul hidup. Jika cost nya lebih besar dari cost simpul solusi, maka matikan simpul tersebut.
5. Jika simpul i bukan simpul solusi, maka bangkitkan semua anak-anaknya. Jika i tidak mempunyai anak, kembali ke langkah 2.
6. Untuk setiap anak j dari simpul i, hitung ĉ(j), dan masukkan semua anak-anak tersebut ke dalam Q.
7. Kembali ke langkah 2

Selain itu ada beberapa heuristik yang diterapkan untuk meminimalisasi Langkah-langkah yang dilalui saat proses penyelesaian puzzle.

1. Menentukan apakah puzzle dapat diselesaikan

Karena terdapat 16! Kemungkinan penempatan tile pada puzzle, dan hanya setengah dari jumlah tersebut yang mempunyai solusi sehingga, dilakukan kalkulasi nilai goal yang menandakan apakah puzzle tersebut mempunyai solusi.

A picture containing text

Description automatically generated

* KURANG(i)

Banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i). POSISI(i) = posisi ubin bernomor i pada susunan yang diperiksa

* X

A picture containing shoji, crossword puzzle

Description automatically generated

X bernilai 1 jika sel kosong pada posisi awal berada pada tile yang diarsir.

1. Menghitung cost simpul (ĉ(i))

ĉ(i) = f(i) + g(i)

* f(i) 🡪 jarak dari simpul akar ke simpul i
* g(i) 🡪 jumlah tile yang tidak sesuai dengan simpul tujuan (target)

1. **Kode Program**

Implementasi program ini memanfaatkan Bahasa pemrograman Java, sehingga dibentuk beberapa kelas untuk memudahkan representasi simpul, puzzle, pembacaan input dan algoritma penyelesaian puzzle menggunakan *branch and bound.*

**Kelas Puzzle (Puzzle.java)**

import java.util.\*;

public class Puzzle{

Integer cost;

ArrayList<Integer> puzzle;

// constructor

Puzzle(){

this.puzzle = new ArrayList<Integer>();

}

Puzzle(Puzzle p){

this.cost = p.cost;

this.puzzle = new ArrayList<Integer>(p.puzzle);

}

// getter & setter

public void setPuzzle(ArrayList<Integer> a){

this.puzzle = new ArrayList<Integer>(a);

}

public void setCost(Integer cost) {

this.cost = cost;

}

public Integer getCost(){

return cost;

}

public Boolean targetReached() {

return this.g().equals(0);

}

// menampilkan puzzle

public void displayPuzzle() {

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

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

System.out.print("[");

if ( this.puzzle.get((4\*i+j)).equals(16)){

System.out.print("-");

}

else{

System.out.print(this.puzzle.get((4\*i+j)));

}

System.out.print("]");

}

System.out.println();

}

}

// menampilkan nilai KURANG(i)

public void displayKurangValue() {

for (int i = 1; i <= this.puzzle.size(); i++){

System.out.print("Kurang(" + i + ") = " + kurang(this.puzzle, i) + "\n");

}

}

// kalkulasi nilai g(i)

public Integer g() {

Integer res = 0;

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

if (!(this.puzzle.get(i).equals(i+1)) && !(this.puzzle.get(i).equals(16))){

res++;

}

}

return res;

}

public Boolean isSolvable(){

return ((reachable(this.puzzle)%2) == 0);

}

// memeriksa nilai sigma(kurang(i)) + X

public static int reachable(ArrayList<Integer> puzzle){

Integer reachable = 0;

for (int i = 0; i < puzzle.size(); i++){

if (!puzzle.get(i).equals(0)){

reachable += kurang(puzzle, i+1);

}

}

return reachable + getX(puzzle);

}

// kalkulasi nilai KURANG(i)

public static int kurang(ArrayList<Integer> puzzle, Integer i){

Integer idx = puzzle.indexOf(i);

Integer res = 0;

for (int j = idx; j < puzzle.size(); j++){

if (puzzle.get(j) < i){

res++;

}

}

return res;

}

// menentukan nilai X

public static int getX(ArrayList<Integer> puzzle){

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

Collections.addAll(arsir,1,3,4,6,9,11,12,14);

Integer emptyidx = puzzle.indexOf(16);

if (arsir.contains(emptyidx)){

return 1;

}

else{

return 0;

}

}

// menentukan apakah move dapat dilakukan dan tidak redundan

public Boolean isMoveSafe(String direction, String lastMove){

Integer emptyidx = this.puzzle.indexOf(16);

if (direction.equals("down")){

return emptyidx <= 11 && lastMove != "up";

}

else if (direction.equals("up")){

return emptyidx >= 4 && lastMove != "down";

}

else if (direction.equals("left")){

return (emptyidx%4) != 0 && lastMove != "right";

}

else{

return ((emptyidx+1)%4) != 0 && lastMove != "left";

}

}

// mengembalikan puzzle yang sudah digerakkan tile-nya

public ArrayList<Integer> move(String direction){

ArrayList<Integer> temp;

if (direction.equals("down")){

temp = new ArrayList<Integer>(this.down());

return temp;

}

else if (direction.equals("up")){

temp = new ArrayList<Integer>(this.up());

return temp;

}

else if (direction.equals("left")){

temp = new ArrayList<Integer>(this.left());

return temp;

}

else{

temp = new ArrayList<Integer>(this.right());

return temp;

}

}

// FUNGSI PERGERAKAN

public ArrayList<Integer> down() {

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx+4);

return temp;

}

public ArrayList<Integer> up(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx-4);

return temp;

}

public ArrayList<Integer> left(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx-1);

return temp;

}

public ArrayList<Integer> right(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx+1);

return temp;

}

}

// menampilkan nilai KURANG(i)

public void displayKurangValue() {

for (int i = 1; i <= this.puzzle.size(); i++){

System.out.print("Kurang(" + i + ") = " + kurang(this.puzzle, i) + "\n");

}

}

// kalkulasi nilai g(i)

public Integer g() {

Integer res = 0;

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

if (!(this.puzzle.get(i).equals(i+1)) && !(this.puzzle.get(i).equals(16))){

res++;

}

}

return res;

}

public Boolean isSolvable(){

return ((reachable(this.puzzle)%2) == 0);

}

// memeriksa nilai sigma(kurang(i)) + X

public static int reachable(ArrayList<Integer> puzzle){

Integer reachable = 0;

for (int i = 0; i < puzzle.size(); i++){

if (!puzzle.get(i).equals(0)){

reachable += kurang(puzzle, i+1);

}

}

return reachable + getX(puzzle);

}

// kalkulasi nilai KURANG(i)

public static int kurang(ArrayList<Integer> puzzle, Integer i){

Integer idx = puzzle.indexOf(i);

Integer res = 0;

for (int j = idx; j < puzzle.size(); j++){

if (puzzle.get(j) < i){

res++;

}

}

return res;

}

// menentukan nilai X

public static int getX(ArrayList<Integer> puzzle){

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

Collections.addAll(arsir,1,3,4,6,9,11,12,14);

Integer emptyidx = puzzle.indexOf(16);

if (arsir.contains(emptyidx)){

return 1;

}

else{

return 0;

}

}

// menentukan apakah move dapat dilakukan dan tidak redundan

public Boolean isMoveSafe(String direction, String lastMove){

Integer emptyidx = this.puzzle.indexOf(16);

if (direction.equals("down")){

return emptyidx <= 11 && lastMove != "up";

}

else if (direction.equals("up")){

return emptyidx >= 4 && lastMove != "down";

}

else if (direction.equals("left")){

return (emptyidx%4) != 0 && lastMove != "right";

}

else{

return ((emptyidx+1)%4) != 0 && lastMove != "left";

}

}

// mengembalikan puzzle yang sudah digerakkan tile-nya

public ArrayList<Integer> move(String direction){

ArrayList<Integer> temp;

if (direction.equals("down")){

temp = new ArrayList<Integer>(this.down());

return temp;

}

else if (direction.equals("up")){

temp = new ArrayList<Integer>(this.up());

return temp;

}

else if (direction.equals("left")){

temp = new ArrayList<Integer>(this.left());

return temp;

}

else{

temp = new ArrayList<Integer>(this.right());

return temp;

}

}

// FUNGSI PERGERAKAN

public ArrayList<Integer> down() {

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx+4);

return temp;

}

public ArrayList<Integer> up(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx-4);

return temp;

}

public ArrayList<Integer> left(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx-1);

return temp;

}

public ArrayList<Integer> right(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx+1);

return temp;

}

}

Collections.addAll(arsir,1,3,4,6,9,11,12,14);

Integer emptyidx = puzzle.indexOf(16);

if (arsir.contains(emptyidx)){

return 1;

}

else{

return 0;

}

}

// menentukan apakah move dapat dilakukan dan tidak redundan

public Boolean isMoveSafe(String direction, String lastMove){

Integer emptyidx = this.puzzle.indexOf(16);

if (direction.equals("down")){

return emptyidx <= 11 && lastMove != "up";

}

else if (direction.equals("up")){

return emptyidx >= 4 && lastMove != "down";

}

else if (direction.equals("left")){

return (emptyidx%4) != 0 && lastMove != "right";

}

else{

return ((emptyidx+1)%4) != 0 && lastMove != "left";

}

}

// mengembalikan puzzle yang sudah digerakkan tile-nya

public ArrayList<Integer> move(String direction){

ArrayList<Integer> temp;

if (direction.equals("down")){

temp = new ArrayList<Integer>(this.down());

return temp;

}

else if (direction.equals("up")){

temp = new ArrayList<Integer>(this.up());

return temp;

}

else if (direction.equals("left")){

temp = new ArrayList<Integer>(this.left());

return temp;

}

else{

temp = new ArrayList<Integer>(this.right());

return temp;

}

}

// FUNGSI PERGERAKAN

public ArrayList<Integer> down() {

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx+4);

return temp;

}

public ArrayList<Integer> up(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx-4);

return temp;

}

public ArrayList<Integer> left(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx-1);

return temp;

}

public ArrayList<Integer> right(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx+1);

return temp;

}

}

**Kelas Node (Node.java)**

}

// FUNGSI PERGERAKAN

public ArrayList<Integer> down() {

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx+4);

return temp;

}

public ArrayList<Integer> up(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx-4);

return temp;

}

public ArrayList<Integer> left(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx-1);

return temp;

}

public ArrayList<Integer> right(){

Integer emptyidx = this.puzzle.indexOf(16);

ArrayList<Integer> temp = new ArrayList<>(this.puzzle);

Collections.swap(temp, emptyidx, emptyidx+1);

return temp;

}

}

public class Node implements Comparable<Node>{

Node parent;

Puzzle puzzle;

Integer level;

String lastMove;

//ctor dan cctor

Node (Puzzle p,Node parent, Integer level, String lastMove){

this.puzzle = p;

this.parent = parent;

this.level = level;

this.lastMove = lastMove;

}

Node (Node n){

this.puzzle = n.puzzle;

this.parent = n.parent;

this.level = n.level;

this.lastMove = n.lastMove;

}

// getter

public Puzzle getPuzzle(){

return this.puzzle;

}

// menampilkan path solusi

public static void printPath(Node root){

if (root == null){

return;

}

printPath(root.parent);

System.out.println("------------");

root.puzzle.displayPuzzle();

System.out.println("------------");

}

// impelementasi PrioQueue, berdasarkan cost paling minimal

@Override

public int compareTo(Node o) {

if (puzzle.getCost() == null || o.puzzle.getCost() == null){

return 0;

}

return puzzle.getCost().compareTo(o.puzzle.getCost());

}

@Override

public String toString(){

return Integer.toString(this.getPuzzle().g());

}

}

**Main Program**

import java.util.ArrayList;

import java.util.Iterator;

import java.util.PriorityQueue;

import java.util.Queue;

import java.util.Scanner;

import java.util.concurrent.TimeUnit;

public class Main {

public static void main(String[] args) {

Integer simpulDibangkitkan = 0;

Queue<Node> pq = new PriorityQueue<>();

Puzzle puzzle = new Puzzle();

Puzzle target = new Puzzle();

ReadText fileReader1 = new ReadText();

ReadText fileReader2 = new ReadText();

Scanner scanner = new Scanner(System.in);

System.out.println("Enter file name with puzzle: ");

String filename = scanner.nextLine();

fileReader1.readPuzzleFile(filename);

fileReader2.readPuzzleFile("target.txt");

puzzle.setPuzzle(fileReader1.getHasil());

target.setPuzzle(fileReader2.getHasil());

puzzle.setCost(puzzle.g());

Node root = new Node(puzzle, null, 0,"None");

pq.add(root);

System.out.println();

System.out.println("Initial State: ");

puzzle.displayPuzzle();

System.out.println();

puzzle.displayKurangValue();

System.out.println("Goal value: " + Puzzle.reachable(puzzle.puzzle));

System.out.println();

System.out.println("Path to solution: ");

if (puzzle.isSolvable()){

// handle exception out of memory

// terjadi kalo puzzle nya rumit

try{

Long startTime = System.nanoTime();

while (pq.size() > 0){

Node minNode = new Node(pq.remove());

// goal state tercapai

if (minNode.getPuzzle().targetReached()){

Node.printPath(minNode);

Long endTime = System.nanoTime();

Long totalTime = endTime-startTime;

System.out.println("Jumlah total simpul dibangkitkan: " + simpulDibangkitkan);

System.out.println("Algorithm total runtime: " + TimeUnit.NANOSECONDS.toMillis(totalTime) + "ms");

return;

}

String moves[] = {"up","down","right","left"};

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

// memeriksa apakah move valid dan tidak redundan

// cth. left-right

if (minNode.getPuzzle().isMoveSafe(moves[i],minNode.lastMove)){

Puzzle tempPuzzle = new Puzzle(minNode.getPuzzle());

tempPuzzle.setPuzzle(minNode.getPuzzle().move(moves[i]));

// minNode.level + 1 menandakan jarak dari root ke node

tempPuzzle.setCost(tempPuzzle.g() + minNode.level + 1);

Node child = new Node(tempPuzzle,minNode,minNode.level + 1,moves[i]);

simpulDibangkitkan++;

pq.add(child);

}

}

}

}

catch(OutOfMemoryError e){

System.out.println("Puzzle terlalu rumit");

}

}

else{

System.out.println("Puzzle tidak dapat diselesaikan");

}

}

}

;

if (puzzle.isSolvable()){

// handle exception out of memory

// terjadi kalo puzzle nya rumit

try{

Long startTime = System.nanoTime();

while (pq.size() > 0){

Node minNode = new Node(pq.remove());

// goal state tercapai

if (minNode.getPuzzle().targetReached()){

Node.printPath(minNode);

Long endTime = System.nanoTime();

Long totalTime = endTime-startTime;

System.out.println("Jumlah total simpul dibangkitkan: " + simpulDibangkitkan);

System.out.println("Algorithm total runtime: " + TimeUnit.NANO SECONDS.toMillis(totalTime) + "ms");

return;

}

String moves[] = {"up","down","right","left"};

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

// memeriksa apakah move valid dan tidak redundan

// cth. left-right

if (minNode.getPuzzle().isMoveSafe(moves[i],minNode.lastMove)){

Puzzle tempPuzzle = new Puzzle(minNode.getPuzzle());

tempPuzzle.setPuzzle(minNode.getPuzzle().move(moves[i]));

// minNode.level + 1 menandakan jarak dari root ke node

tempPuzzle.setCost(tempPuzzle.g() + minNode.level + 1);

Node child = new Node(tempPuzzle,minNode,minNode.level + 1,moves[i]);

simpulDibangkitkan++;

pq.add(child);

}

}

}

}

catch(OutOfMemoryError e){

System.out.println("Puzzle terlalu rumit");

}

}

else{

System.out.println("Puzzle tidak dapat diselesaikan");

}

}

}

;

1. **Hasil pengujian Program**

|  |  |
| --- | --- |
| Puzzle yang diuji | Bukti Screenshot Output Program |
| Text  Description automatically generated | Text  Description automatically generated with low confidenceText  Description automatically generated with medium confidence |
| Text  Description automatically generated | Text  Description automatically generated with low confidence |
| Text  Description automatically generated | A picture containing calendar  Description automatically generatedA picture containing text  Description automatically generated |
| Text  Description automatically generated | Text  Description automatically generatedA picture containing diagram  Description automatically generated |
| Text  Description automatically generated | A picture containing text  Description automatically generated |

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

Link Repository:

https://github.com/geraldakbar/Tucil2Stima-13520143