import java.util.Collection;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.PriorityQueue;

import java.util.Scanner;

import java.util.Stack;

import java.util.\*;

import java.io.\*;

public class Solver {

int count = 0;

Integer width;

Integer height;

Node myHead;

HashSet<Integer> visited;

HashSet<Block> blocksInitial = new HashSet<Block>();

HashSet<Block> finalblocks = new HashSet<Block>();

static String init;

static String goal;

boolean finished;

Comparator<Node> comparator = new TrayComparator();

PriorityQueue<Node> priority;

int maxheight = 0; // same as height

int maxwidth = 0; // same as width

public Solver(String startfile, String endfile) {

try {

/\*\*

\* Change the directory and file name for the new File object with

\* each call to readInt. Also you need the try catch

\*/

comparator = new TrayComparator();

blocksInitial = new HashSet<Block>();

finalblocks = new HashSet<Block>();

Scanner scan = new Scanner(new File(init)); // change it here

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

int count = 0;

while (scan.hasNextLine()) {

String line = scan.nextLine();

String[] intsInLine = line.split("\\s+");

if (count == 0) {

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

for (int i = 0; i < intsInLine.length; i++) {

try {

int toAdd = Integer.parseInt(intsInLine[i]);

if (i == 0) {

maxheight = toAdd;

} else if (i == 1) {

maxwidth = toAdd;

}

temp.add(toAdd);

} catch(NumberFormatException e) {

System.out.println("Invalid Init and/or goal file");

return;

}

}

if (temp.size() != 2) {

System.out.println("Invalid Init and/or goal file");

return;

}

allInts.add(temp);

} else {

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

for (int i = 0; i < intsInLine.length; i++) {

try {

int toAdd = Integer.parseInt(intsInLine[i]);

if (i == 0 || i == 2) {

if (toAdd >= maxheight) {

System.out.println("Invalid Init and/or goal file");

return;

}

} else if (i == 1 || i == 3) {

if (toAdd >= maxwidth) {

System.out.println("Invalid Init and/or goal file");

return;

}

}

temp.add(toAdd);

}catch (NumberFormatException e) {

System.out.println("Invalid Init and/or goal file");

return;

}

}

if (temp.size() != 4) {

System.out.println("Invalid Init and/or goal file");

return;

}

allInts.add(temp);

}

count++;

}

height = allInts.get(0).get(1);

width = allInts.get(0).get(0);

visited = new HashSet<Integer>();

for (int k = 1; k < allInts.size(); k++) {

blocksInitial.add(new Block(allInts.get(k)));

}

Scanner scanEnd = new Scanner(new File(goal));

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

int countEnd = 0;

while (scanEnd.hasNextLine()) {

String line = scanEnd.nextLine();

String[] intsInLine = line.split("\\s+");

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

for (int i = 0; i < intsInLine.length; i++) {

try {

temp.add(Integer.parseInt(intsInLine[i]));

} catch (NumberFormatException e) {

System.out.println("Invalid Init and/or goal file");

return;

}

}

if (temp.size() != 4) {

System.out.println("Invalid Init and/or goal file");

return;

}

allIntsEnd.add(temp);

}

countEnd++;

/\*\*

\* The For Loop underneath is only used to print the contents of the

\* ArrayList

\*/

scan.close();

for (int k = 0; k < allIntsEnd.size(); k++) {

finalblocks.add(new Block(allIntsEnd.get(k)));

}

priority = new PriorityQueue<Node>(blocksInitial.size() \* 4,

comparator);

finished = false;

visited.add(getHashCode(blocksInitial));

myHead = new Node(null, new ArrayList<Integer>(), blocksInitial);

priority.add(myHead);

startChildren();

} catch (FileNotFoundException e) {

System.out.println("Invalid Init and/or goal file");

}

}

public boolean checkFinal(HashSet<Block> compare, Node start) {

int count = 0;

if (finished == false) {

for (Block f : finalblocks) {

count += 1;

if (!compare.contains(f)) {

return false;

}

}

finished = true;

Stack<ArrayList<Integer>> toPrint = new Stack<ArrayList<Integer>>();

Node temp = start;

while (temp != myHead && temp != null) {

toPrint.add(temp.move);

temp = temp.myPrev;

}

while (!toPrint.isEmpty()) {

ArrayList<Integer> print = toPrint.pop();

if (!print.isEmpty()) {

for (Integer i : print) {

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

}

System.out.println();

}

}

return true;

}

return false;

}

private class Node {

String[][] myMatrix;

ArrayList<Integer> move;

ArrayList<Node> myChildren;

Node myPrev;

HashSet<Block> blocks = new HashSet<Block>();

public Node(Node parent, ArrayList<Integer> m, HashSet<Block> b) {

blocks = b;

move = m;

myPrev = parent;

myChildren = new ArrayList<Node>();

myMatrix = new String[width][height];

checkFinal(blocks, this);

if (!finished) {

for (Block z : blocks) {

addBlock("taken", z);

}

}

}

public void addBlock(String blockname, Block b) {

for (int k = b.coords.get(0); k <= b.coords.get(2); k++) {

for (int t = b.coords.get(1); t <= b.coords.get(3); t++) {

myMatrix[k][t] = blockname;

}

}

}

public ArrayList<Integer> moveLeft(ArrayList<Integer> coords) {

int h = Math.abs(coords.get(3) - coords.get(1)) + 1;

int x = coords.get(0) - 1;

if (x >= 0) {

for (int k = coords.get(1); k < coords.get(1) + h; k++) {

if (myMatrix[x][k] != null) {

return null;

}

}

} else {

return null;

}

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

t.add(coords.get(0));

t.add(coords.get(1));

t.add(coords.get(0) - 1);

t.add(coords.get(1));

return t;

}

public ArrayList<Integer> moveRight(ArrayList<Integer> coords) {

int h = Math.abs(coords.get(3) - coords.get(1)) + 1;

int x = coords.get(2) + 1;

if (x < width) {

for (int k = coords.get(1); k < coords.get(1) + h; k++) {

if (myMatrix[x][k] != null) {

return null;

}

}

} else {

return null;

}

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

t.add(coords.get(0));

t.add(coords.get(1));

t.add(coords.get(0) + 1);

t.add(coords.get(1));

return t;

}

public ArrayList<Integer> moveUp(ArrayList<Integer> coords) {

int w = Math.abs(coords.get(2) - coords.get(0)) + 1;

int y = coords.get(1) - 1;

if (y >= 0) {

for (int k = coords.get(0); k < coords.get(0) + w; k++) {

if (myMatrix[k][y] != null) {

return null;

}

}

} else {

return null;

}

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

t.add(coords.get(0));

t.add(coords.get(1));

t.add(coords.get(0));

t.add(coords.get(1) - 1);

return t;

}

public ArrayList<Integer> moveDown(ArrayList<Integer> coords) {

int w = Math.abs(coords.get(2) - coords.get(0)) + 1;

int y = coords.get(3) + 1;

if (y < height) {

for (int k = coords.get(0); k < coords.get(0) + w; k++) {

if (myMatrix[k][y] != null) {

return null;

}

}

} else {

return null;

}

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

t.add(coords.get(0));

t.add(coords.get(1));

t.add(coords.get(0));

t.add(coords.get(1) + 1);

return t;

}

public void children() {

for (Block b : blocks) {

ArrayList<Integer> block = b.coords;

if (finished == false) {

if (moveLeft(block) != null) {

HashSet<Block> n = new HashSet<Block>();

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

n.addAll(blocks);

n.remove(b);

t.add(block.get(0) - 1);

t.add(block.get(1));

t.add(block.get(2) - 1);

t.add(block.get(3));

n.add(new Block(t));

if (!visited.contains(getHashCode(n))) {

visited.add(getHashCode(n));

Node child1 = new Node(this, moveLeft(block), n);

priority.add(child1);

}

}

}

if (finished == false) {

if (moveRight(block) != null) {

HashSet<Block> n = new HashSet<Block>();

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

n.addAll(blocks);

n.remove(b);

t.add(block.get(0) + 1);

t.add(block.get(1));

t.add(block.get(2) + 1);

t.add(block.get(3));

n.add(new Block(t));

if (!visited.contains(getHashCode(n))) {

visited.add(getHashCode(n));

Node child2 = new Node(this, moveRight(block), n);

priority.add(child2);

}

}

}

if (finished == false) {

if (moveUp(block) != null) {

HashSet<Block> n = new HashSet<Block>();

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

n.addAll(blocks);

n.remove(b);

t.add(block.get(0));

t.add(block.get(1) - 1);

t.add(block.get(2));

t.add(block.get(3) - 1);

n.add(new Block(t));

if (!visited.contains(getHashCode(n))) {

visited.add(getHashCode(n));

Node child3 = new Node(this, moveUp(block), n);

priority.add(child3);

}

}

}

if (finished == false) {

if (moveDown(block) != null) {

HashSet<Block> n = new HashSet<Block>();

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

n.addAll(blocks);

n.remove(b);

t.add(block.get(0));

t.add(block.get(1) + 1);

t.add(block.get(2));

t.add(block.get(3) + 1);

n.add(new Block(t));

if (!visited.contains(getHashCode(n))) {

visited.add(getHashCode(n));

Node child4 = new Node(this, moveDown(block), n);

priority.add(child4);

}

}

}

}

}

}

public void startChildren() {

while (finished == false && !priority.isEmpty()) {

Node popped = priority.remove();

popped.children();

}

}

public Integer getHashCode(HashSet<Block> toHash) {

Integer toReturn = 1;

for (Block s : toHash) {

toReturn \*= 57;

toReturn += s.hashCode();

}

return toReturn;

}

private static boolean fileExists(String root, String filename,

boolean isInitFile) {

File file = new File(root, filename);

if (file.exists()) {

if (isInitFile == true) {

init = root + "/" + filename;

return true;

} else {

goal = root + "/" + filename;

return true;

}

} else {

boolean exists = false;

file = new File(root);

File[] files = file.listFiles();

if (files != null) {

for (File f : files) {

if (exists) {

return exists;

} else if (f.isDirectory()) {

exists = (false || fileExists(f.getPath(), filename,

isInitFile));

}

}

}

return exists;

}

}

private class TrayComparator implements Comparator<Node> {

int num = 0;

public int compare(Node x, Node y) {

if (Heuristic(x.blocks) > Heuristic(y.blocks)) {

return 1;

} else if (Heuristic(x.blocks) < Heuristic(y.blocks)) {

return -1;

} else {

return 0;

}

}

public int Heuristic(HashSet<Block> hash) {

count -= 1;

if (blocksInitial.size() > 200) {

return count;

}

int min = Integer.MAX\_VALUE;

int d = 0;

for (Block b : finalblocks) {

for (Block c : hash) {

if (c.getSize().equals(b.getSize())) {

d = (int) (Math.pow(Math.abs(b.coords.get(1) - c.coords.get(1)), 2) +

Math.pow(Math.abs(b.coords.get(0) - c.coords.get(0)),2));

if (d < min) {

min = d;

}

}

}

}

return d;

}

}

public static void main(String[] args) {

if (args.length == 2) {

if (fileExists("./", args[0], true)

&& fileExists("./", args[1], false)) {

Solver s = new Solver(init, goal);

} else {

System.out.println("Invalid Init and/or goal file");

}

} else {

System.out.println("Invalid Init and/or goal file");

}

return;

}

}