import java.util.\*;

/\*\*

\* breadth-first search algorithm to solve a 3x3 sliding puzzle.

\*

\* @author (Jack Delancey B519170)

\* @version (Version 1)

\*/

public class PuzzleSolver

{

// instance variables

Integer[] goalArray; // the goal state

Integer[] availArray; // array of available moves

Integer[] moveArray; // temporary array to store values while it's switching them

Integer[] stateArray; // the current state of the puzzle, originally populated with start state

Integer[] newArray;

ArrayList<Integer[]> prevArray; // list of arrays that have been searched already

ArrayList<Integer[]> queueArray; // queue of arrays to be searched through

Queue<Integer[]> results;// queue of results from searching nodes

boolean stop; // stops the search

/\*\*

\* Constructor for objects of class PuzzleSolver

\*/

public PuzzleSolver()

{

goalArray = new Integer[9]; // must be populated with 9 values

availArray = new Integer[4]; // must be populated with 4 values

moveArray = new Integer[9]; // must be populated with 9 values

stateArray = new Integer[9]; // must be populated with 9 values

newArray = new Integer[9]; // must be populated with 9 values

prevArray = new ArrayList<Integer[]>();

queueArray = new ArrayList<Integer[]>();

results = new LinkedList<Integer[]>();

stop = false; // inital state of stop is false

}

public void puzzle(int a, int b, int c, int d, int e, int f, int g, int h, int i){

//the current state of the state array/sliding puzzle, this is continuously updated after each iteration

stateArray = new Integer[9];

//positions within the sliding puzzle/array

stateArray[0] = a; //top left

stateArray[1] = b; //top middle

stateArray[2] = c; //top right

stateArray[3] = d; //middle left

stateArray[4] = e; //middle middle

stateArray[5] = f; //middle right

stateArray[6] = g; //bottom left

stateArray[7] = h; //bottom middle

stateArray[8] = i; //bottom right

results.add(stateArray);

}

public void result(int a, int b, int c, int d, int e, int f, int g, int h, int i){

//this array of populated with the goal state so the algorithm knows when to stop

//positions within the sliding puzzle/array(all arrays have same lay out)

goalArray[0] = a; //top left

goalArray[1] = b; //top middle

goalArray[2] = c; //top right

goalArray[3] = d; //middle left

goalArray[4] = e; //middle middle

goalArray[5] = f; //middle right

goalArray[6] = g; //bottom left

goalArray[7] = h; //bottom middle

goalArray[8] = i; //bottom right

}

public void swap(){

int number = 0;

int i = 0;

i = 0;

for (Integer n : stateArray){

//loops through stateArray

newArray[i] = n;

i = i + 1;

}

if (Arrays.asList(stateArray).indexOf(0) == 0){

//list of available moves

//-1 indicates the movement cant be made

availArray[0] = -1; //up however the move cant actually be made

availArray[1] = 3; //can move down to index 3

availArray[2] = -1; //left however the move cant be made

availArray[3] = 1; //can move right to index 1

}

if (Arrays.asList(stateArray).indexOf(0) == 1){

//list of available moves

//-1 indicates the movement cant be made

availArray[0] = -1; // up, move cant be made

availArray[1] = 4; //down to index 4

availArray[2] = 0; //left to index 0

availArray[3] = 2; //right to index 2

}

if (Arrays.asList(stateArray).indexOf(0) == 2){

//list of available moves

//-1 indicates the movement cant be made

availArray[0] = -1; //up, move cant be made

availArray[1] = 5; //down to index 5

availArray[2] = 1; //left to index 1

availArray[3] = -1; //right, move cant be made

}

if (Arrays.asList(stateArray).indexOf(0) == 3){

//list of available moves

//-1 indicates the movement cant be made

availArray[0] = 0; //up to index 0

availArray[1] = 6; //down to index 6

availArray[2] = -1; //left, move cant be made

availArray[3] = 4; //right to index 4

}

if (Arrays.asList(stateArray).indexOf(0) == 4){

//list of available moves

//-1 indicates the movement cant be made

availArray[0] = 1; //up to index 1

availArray[1] = 7; //down to index 7

availArray[2] = 3; //left to index 3

availArray[3] = 5; //right to index 5

}

if (Arrays.asList(stateArray).indexOf(0) == 5){

//list of available moves

//-1 indicates the movement cant be made

availArray[0] = 2; //up to index 2

availArray[1] = 8; //down to index 8

availArray[2] = 4; //left to index 4

availArray[3] = -1; //right, move cant be made

}

if (Arrays.asList(stateArray).indexOf(0) == 6){

//list of available moves

//-1 indicates the movement cant be made

availArray[0] = 3; //up to index 3

availArray[1] = -1; //down, move cant be made

availArray[2] = -1; //left, move cant be made

availArray[3] = 7; //right to index 7

}

if (Arrays.asList(stateArray).indexOf(0) == 7){

//list of available moves

//-1 indicates the movement cant be made

availArray[0] = 4; //up to index 4

availArray[1] = -1; //down, move cant be made

availArray[2] = 6; //left to index 6

availArray[3] = 8; //right to index 8

}

if (Arrays.asList(stateArray).indexOf(0) == 8){

//list of available moves

//-1 indicates the movement cant be made

availArray[0] = 5; //up to index 5

availArray[1] = -1; //down, move cant be made

availArray[2] = 7; //left to index 7

availArray[3] = -1; //right, move cant be made

}

if(!(availArray[0] == -1)){

//if there is an available move up

i = 0;

for (Integer n : newArray){

moveArray[i] = n;

i = i + 1;

}

number = moveArray[availArray[0]];

moveArray[availArray[0] + 3] = number;

moveArray[availArray[0]] = 0;

i = 0;

stateArray = new Integer[9];

for (Integer n : moveArray){

stateArray[i] = n;

i = i + 1;

}

results.add(stateArray);

}

if(!(availArray[1] == -1)){

//if there is an available move down

i = 0;

for (Integer n : newArray){

moveArray[i] = n;

i = i + 1;

}

number = moveArray[availArray[1]];

moveArray[availArray[1] - 3] = number;

moveArray[availArray[1]] = 0;

i = 0;

stateArray = new Integer[9];

for (Integer n : moveArray){

stateArray[i] = n;

i = i + 1;

}

results.add(stateArray);

}

if(!(availArray[2] == -1)){

//if there is an available move left

i = 0;

for (Integer n : newArray){

moveArray[i] = n;

i = i + 1;

}

number = moveArray[availArray[2]];

moveArray[availArray[2] + 1] = number;

moveArray[availArray[2]] = 0;

i = 0;

stateArray = new Integer[9];

for (Integer n : moveArray){

stateArray[i] = n;

i = i + 1;

}

results.add(stateArray);

}

if(!(availArray[3] == -1)){

//if there is an available move right

i = 0;

for (Integer n : newArray){

moveArray[i] = n;

i = i + 1;

}

number = moveArray[availArray[3]];

moveArray[availArray[3] - 1] = number;

moveArray[availArray[3]] = 0;

i = 0;

stateArray = new Integer[9];

for (Integer n : moveArray){

stateArray[i] = n;

i = i + 1;

}

results.add(stateArray);

}

}

public void solve(){

int x = 0;

int j = 0;

while(!stop){

//starts with a clear array

queueArray.clear();

for (Integer[] n : results){

//populates with results to search through

queueArray.add(n);

}

for (Integer[] entry : queueArray){

System.out.println("q: " + Arrays.toString(entry));

}

for (Integer[] entry : queueArray){

int i = 0;

stateArray = new Integer[9];

for (Integer n : entry){

stateArray[i] = n;

i = i + 1;

}

if(!(Arrays.toString(stateArray).equals(Arrays.toString(goalArray)))){

if (prevArray.contains(results.element())){

results.remove();

}

else {

System.out.println("Entry: " + Arrays.toString(stateArray));

this.swap();

i = 0;

stateArray = new Integer[9];

for (Integer n : entry){

stateArray[i] = n;

i = i + 1;

}

prevArray.add(stateArray);

x = x + 1;

System.out.println(x);

results.remove();

}

}

else{

//what is finally displayed once the puzzle is solved

stop = true;

x = x + 1;

System.out.println(stop);

System.out.println(Arrays.toString(stateArray));

System.out.println("Found in: " + (x - 1));

break;

}

}

j = j + 1;

}

}

}