Universiteti Politeknik i Tiranës

Fakulteti i Teknologjisë së Informacionit

Dega: Inxhinieri Informatike

Grupi: III-B

Viti akademik 2024-2025

Punë laboratori nr. 2

Lënda: Algoritmike dhe programim i avancuar

Punoi: Piro Gjidhima

Pranoi:Msc Alba Haveriku

Klasa **Board.java**

```
import edu.princeton.cs.algs4.In;

import java.util.ArrayList;

public class Board {
    private final int[][] tiles;
    protected final int distance;

    public Board(int[][] tiles) {
        this.tiles = tiles;
        distance = manhattan();
    }

    public String toString() {
        String view = size() + " \n";


        for (int i = 0; i < size(); i++) {
            for (int j = 0; j < size(); j++) {
                view += tileAt(i, j) + " ";
            }
            view += "\n";
        }
        return view;
    }

    public int tileAt(int row, int col) {

        if (!(0 <= row && row < size()) || !(0 <= col && col < size()))
            throw new IllegalArgumentException();

        return this.tiles[row][col];
    }

    public int size() {
        return this.tiles.length;
    }

    public int hamming()

        int distance = 0;

        for (int i = 0, k = 1; i < size(); i++) {
            for (int j = 0; j < size(); j++, k++) {
                if (tileAt(i, j) == 0) {
```

```
                k--;
                continue;
            }
            if (tileAt(i, j) != k)
                distance++;
        }

    }
    return distance;
}

public int manhattan() {

    int distance = 0;

    for (int i = 0; i < size(); i++) {
        for (int j = 0; j < size(); j++) {
            int current = tileAt(i, j);
            if (current != 0) {
                int row = (current - 1) / size();
                int col = (current - 1) % size();
                distance += Math.abs(i - row) + Math.abs(j - col);
            }
        }
    }
    return distance;
}

public boolean isGoal() {

    for (int i = 0, k = 1; i < size(); i++) {
        for (int j = 0; j < size(); j++, k++) {
            if (tileAt(i, j) == tileAt(size() - 1, size() - 1))
                continue;

            if (tileAt(i, j) != k)
                return false;
        }

    }
    return true;
}

public boolean equals(Object y) {

    if (y == null)
        throw new NullPointerException();
        if (y.getClass() != this.getClass())
            throw new IllegalArgumentException();

        final Board temp = (Board) y;

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

            for (int i = 0; i < size(); i++) {
                for (int j = 0; j < size(); j++) {
                    if (tileAt(i, j) != temp.tileAt(i, j))
                        return false;
                }
            }
            return true;
        }
        return false;
    }

    public Iterable<Board> neighbors() {
        ArrayList<Board> boards = new ArrayList<>();
        int row = size() - 1, col = size() - 1;

        for (int i = 0; i < size(); i++) {
            for (int j = 0; j < size(); j++) {
                if (tileAt(i, j) == 0) {
                    row = i;
                    col = j;
                    break;
                }
            }
        }

        // Lart
        if (row - 1 >= 0) {

            int[][] copy = new int[size()][size()];
            for (int i = 0; i < size(); i++) {
                for (int j = 0; j < size(); j++) {
                    copy[i][j] = tileAt(i, j);
                }
            }
            Board tempBoard = new Board(copy);
            tempBoard.tiles[row][col] = tileAt(row - 1, col);
            tempBoard.tiles[row - 1][col] = 0;
            boards.add(tempBoard);
```

```
}

// Poshte
if (row + 1 <= size() - 1) {
   int[][] copy = new int[size()][size()];
   for (int i = 0; i < size(); i++) {
      for (int j = 0; j < size(); j++) {
         copy[i][j] = tileAt(i, j);
      }
   }
   Board tempBoard = new Board(copy);
   tempBoard.tiles[row][col] = tileAt(row + 1, col);
   tempBoard.tiles[row + 1][col] = 0;
   boards.add(tempBoard);
}

// Majtas
if (col - 1 >= 0) {
   int[][] copy = new int[size()][size()];
   for (int i = 0; i < size(); i++) {
      for (int j = 0; j < size(); j++) {
         copy[i][j] = tileAt(i, j);
      }
   }
   Board tempBoard = new Board(copy);
   tempBoard.tiles[row][col] = tileAt(row, col - 1);
   tempBoard.tiles[row][col - 1] = 0;
   boards.add(tempBoard);

}

// Djathtas
if (col + 1 <= size() - 1) {

   int[][] copy = new int[size()][size()];
   for (int i = 0; i < size(); i++) {
      for (int j = 0; j < size(); j++) {
         copy[i][j] = tileAt(i, j);
      }
   }
   Board tempBoard = new Board(copy);
   tempBoard.tiles[row][col] = tileAt(row, col + 1);
   tempBoard.tiles[row][col + 1] = 0;
   boards.add(tempBoard);
}

return boards;
```

```
    }

public boolean isSolvable() {

    int[] temp = new int[size() * size()];
    int index = 0;
    int sum = 0;

    for (int i = 0; i < size(); i++) {
        for (int j = 0; j < size(); j++) {
            if (tileAt(i, j) == 0) {
                sum += i;
            }
            temp[index++] = tileAt(i, j);
        }
    }

    int inversions = 0;
    int n = temp.length;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (temp[i] != 0 && temp[j] != 0 && temp[i] > temp[j]) {
                inversions++;
            }
        }
    }

    if (size() % 2 == 1) {
        if (inversions % 2 == 0)
            return true;
        else
            return false;
    }
    else {

        sum += inversions;
        if (sum % 2 == 1) {
            return true;
        }
        else
            return false;
    }
}
```

```
    public static void main(String[] args) {

        In in = new In("puzzle50.txt");
        int n = in.readInt();
        int[][] tiles = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                tiles[i][j] = in.readInt();
            }
        }

        Board initial = new Board(tiles);
        System.out.println(initial.isSolvable());

    }
}
```

Klasa **Solver.java**

```
import edu.princeton.cs.algs4.In;
import edu.princeton.cs.algs4.MinPQ;

import java.util.LinkedList;

public class Solver {
    private final int moves;
    private final Iterable<Board> solutionPath;

    private class SearchNode implements Comparable<SearchNode> {
        private final Board current;
        private final SearchNode prev;
        private final int moves;
        private final int priority;

        public SearchNode(Board current, SearchNode prev) {
            this.current = current;
            this.prev = prev;
            this.moves = (prev == null) ? 0 : prev.moves + 1;
            this.priority = this.moves + current.distance;
        }

        @Override
        public int compareTo(SearchNode other) {
            return Integer.compare(this.priority, other.priority);
        }
    }
```

```
public Solver(Board initial) {
    if (initial == null || !initial.isSolvable()) {
        throw new IllegalArgumentException();
    }
    MinPQ<SearchNode> queue = new MinPQ<>();
    queue.insert(new SearchNode(initial, null));
    SearchNode goal = null;

    while (!queue.isEmpty()) {

        SearchNode node = queue.delMin();

        if (node.current.isGoal()) {
            goal = node;
            break;
        }

        for (Board neighbor : node.current.neighbors()) {
            if (node.prev != null && neighbor.equals(node.prev.current)) {
                continue;
            }
            queue.insert(new SearchNode(neighbor, node));
        }

    }


    assert goal != null;// Nuk mund te jete null nese board.isSolvable()==true

    this.moves = goal.moves;

    LinkedList<Board> path = new LinkedList<>();

    while (goal != null) {
        path.addFirst(goal.current);
        goal = goal.prev;
    }

    this.solutionPath = path;
}


public int moves() {
    return this.moves;
}

public Iterable<Board> solution() {
        return solutionPath;

    }

    public static void main(String[] args) {

        In in = new In("puzzle50.txt");
        int n = in.readInt();
        int[][] tiles = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                tiles[i][j] = in.readInt();
            }
        }

        Board initial = new Board(tiles);

        Solver solver = new Solver(initial);

        for (Board board : solver.solution()) {
            System.out.println(board);
        }

    }
}
```