

Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации

государственное бюджетное образовательное учреждение

высшего профессионального образования

ордена Трудового Красного Знамени

“Московский технический университет связи и информатики”

Лабораторная работа №3 по дисциплине

“ Структуры и алгоритмы обработки данных”

Выполнил студент

Группы БВТ1902

Ахмедов Х.М.

Москва 2021

Оглавление

1. Задание на лабораторную работу	3
2. Листинг программы	4
3. Вывод.....	21

1. Задание на лабораторную работу

Задание 1

Реализовать методы поиска подстроки в строке. Добавить возможность ввода строки и подстроки с клавиатуры. Предусмотреть возможность существования пробела. Реализовать возможность выбора опции чувствительности или нечувствительности к регистру. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования.

Алгоритмы:

1. Кнута-Морриса-Пратта
2. Упрощенный Бойера-Мура

Задание 2 «Пятнашки»

Игра в 15, пятнашки, такен — популярная головоломка, придуманная в 1878 году Ноем Чепмэном. Она представляет собой набор одинаковых квадратных костяшек с нанесёнными числами, заключённых в квадратную коробку. Длина стороны коробки в четыре раза больше длины стороны костяшек для набора из 15 элементов, соответственно в коробке остаётся незаполненным одно квадратное поле. Цель игры — перемещая костяшки по коробке, добиться упорядочивания их по номерам, желательно сделав как можно меньше перемещений.



На рисунках выше изображены различные позиции элементов в задаче:

1. Левый рисунок — одна из возможных начальных позиций элементов.
2. Средний рисунок — одна из «нерешаемых» позиций.
3. Правый рисунок — позиция, где все элементы расставлены в правильном порядке.

Задача: написать программу, определяющую, является ли данное расположение «решаемым», то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

Входные данные: массив чисел, представляющий собой расстановку в порядке «слева направо, сверху вниз». Число 0 обозначает пустое поле. Например, массив [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0] представляет собой «решенную» позицию элементов.

Выходные данные: если решения нет, то функция должна вернуть пустой массив []. Если решение есть, то необходимо представить решение — для каждого шага записывается номер передвигаемого на данном шаге элемента.

Например, для начального расположения элементов [1, 2, 3, 4, 5, 6, 7, 8, 13, 9, 11, 12, 10, 14, 15, 0] одним из возможных решений будет [15, 14, 10, 13, 9, 10, 14, 15] (последовательность шагов здесь: двигаем 15, двигаем 14, двигаем 10, ..., двигаем 15).

2. Листинг программы

Алгоритм Кнута-Морриса-Пратта

```
package ThirdLab;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

public class Knut_Morris_Pratt {
    /** Failure array */
    private final int[] Pi;
    /** Constructor */
    public Knut_Morris_Pratt(String text, String pattern)
    {
        /** pre construct failure array for a pattern */
        Pi = new int[pattern.length()];
        fail(pattern);
        /** find match */
        int pos = posMatch(text, pattern);
        if (pos == -1)
            System.out.println("\nNo match found");
        else
            System.out.println("\nMatch found at index " + pos);
    }
    /** Failure function for a pattern */
    private void fail(String pat)
    {
        int n = pat.length();
        Pi[0] = 0;
```

```

        for (int j = 1; j < n; j++)
        {
            int i = Pi[j - 1];
            while ((pat.charAt(j) != pat.charAt(i + 1)) && i >= 0)
                i = Pi[i];
            if (pat.charAt(j) == pat.charAt(i + 1))
                Pi[j] = i + 1;
            else
                Pi[j] = -1;
        }
    }
    /** Function to find match for a pattern */
    private int posMatch(String text, String pat)
    {
        int i = 0, j = 0;
        int lens = text.length();
        int lenp = pat.length();
        while (i < lens && j < lenp)
        {
            if (text.charAt(i) == pat.charAt(j))
            {
                i++;
                j++;
            }
            else if (j == 0)
                i++;
            else
                j = Pi[j - 1] + 1;
        }
        return ((j == lenp) ? (i - lenp) : -1);
    }
    /**    */
    public static void main(String[] args) throws IOException
    {
        Scanner scanner = new Scanner(System.in);
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        System.out.println("Knuth Morris Pratt Test\n");
        System.out.println("Введите строку: ");
        String text = br.readLine();
        System.out.println("Введите подстроку: ");
        String pattern = br.readLine();
        System.out.println("Введите чувствительность к регистру(0 - нечувст,
1 - чувст): ");
        int registr = scanner.nextInt();
        if(registr == 0){
            String textLower = text.toLowerCase();
            String patternLower = pattern.toLowerCase();
            Knut_Morris_Pratt kmp = new Knut_Morris_Pratt(textLower,
patternLower);
        }else if(registr == 1){
            Knut_Morris_Pratt kmp = new Knut_Morris_Pratt(text, pattern);
        }
    }
}

```

Алгоритм Бойера-Мура

```

package ThirdLab;

import java.io.BufferedReader;
import java.io.IOException;

```

```

import java.io.InputStreamReader;
import java.util.Scanner;

public class BoyerMoore {
    /** Функция findPattern */
    public void findPattern(String t, String p)
    {
        char[] text = t.toCharArray();
        char[] pattern = p.toCharArray();
        int pos = indexOf(text, pattern);
        if (pos == -1)
            System.out.println("\nNo Match\n");
        else
            System.out.println("Pattern found at position : "+ pos);
    }

    /** Функция для вычисления индекса подстроки шаблона */
    public int indexOf(char[] text, char[] pattern)
    {
        if (pattern.length == 0)
            return 0;
        int[] charTable = makeCharTable(pattern);
        int[] offsetTable = makeOffsetTable(pattern);
        for (int i = pattern.length - 1; i < text.length; i++)
        {
            for (j = pattern.length - 1; pattern[j] == text[i]; --j, --i)
            {
                if (j == 0)
                    return i;

                // i += pattern.length - j; // For naive method
                i += Math.max(offsetTable[pattern.length - 1 - j],
charTable[text[i]]);
            }
            return -1;
        }

        /** Создает таблицу переходов на основе информации о несовпадающих символах */
        private int[] makeCharTable(char[] pattern)
        {
            final int ALPHABET_SIZE = 256;
            int[] table = new int[ALPHABET_SIZE];
            for (int i = 0; i < table.length; ++i)
                table[i] = pattern.length;
            for (int i = 0; i < pattern.length - 1; ++i)
                table[pattern[i]] = pattern.length - 1 - i;
            return table;
        }

        /** Создает таблицу переходов на основе смещения сканирования, при котором возникает несоответствие. */
        private static int[] makeOffsetTable(char[] pattern)
        {
            int[] table = new int[pattern.length];
            int lastPrefixPosition = pattern.length;
            for (int i = pattern.length - 1; i >= 0; --i)
            {
                if (isPrefix(pattern, i + 1))
                    lastPrefixPosition = i + 1;
                table[pattern.length - 1 - i] = lastPrefixPosition - i +
pattern.length - 1;
            }
            for (int i = 0; i < pattern.length - 1; ++i)
            {
                int slen = suffixLength(pattern, i);
                table[slen] = pattern.length - 1 - i + slen;
            }
        }
    }
}

```

```

    }
    return table;
}
/** функция, чтобы проверить, является ли игла [p: end] префиксом
шаблона**/
private static boolean isPrefix(char[] pattern, int p)
{
    for (int i = p, j = 0; i < pattern.length; ++i, ++j)
        if (pattern[i] != pattern[j])
            return false;
    return true;
}
/** функция, возвращающая максимальную длину подстроки, оканчивающейся на
p и являющейся суффиксом **/
private static int suffixLength(char[] pattern, int p)
{
    int len = 0;
    for (int i = p, j = pattern.length - 1; i >= 0 && pattern[i] ==
pattern[j]; --i, --j)
        len += 1;
    return len;
}
/** Main Function **/
public static void main(String[] args) throws IOException
{
    Scanner scanner = new Scanner(System.in);
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
    System.out.println("Boyer Moore Algorithm Test\n");
    System.out.println("Введите строку: ");
    String text = br.readLine();
    System.out.println("Введите подстроку: ");
    String pattern = br.readLine();
    System.out.println("Введите чувствительность к регистру(0 - нечувств,
1 - чувств): ");
    int registr = scanner.nextInt();
    BoyerMoore bm = new BoyerMoore();
    if(registr == 0){
        String textLower = text.toLowerCase();
        String patternLower = pattern.toLowerCase();
        bm.findPattern(textLower, patternLower);
    }else if(registr == 1){
        bm.findPattern(text, pattern);
    }
}
}

```

Пятнашки

```

package ThirdLab.Algorithm_Astar;

import java.util.Collection;
import java.util.LinkedList;
import java.util.List;

/**
 * Реализует алгоритм поиска решения A*.
 */
public class Astar<TState extends State, TRules extends Rules<TState>> {

    /**
     * Применяет алгоритм A* для поиска кратчайшего пути до терминального
     * состоянийот указанного.
     */
}

```

```

*
* @param startState - начальное состояние.
* @return последовательность состояний от заданного до терминального.
*/
public Collection<State> search(TState startState) {
    LinkedList<Integer> close = new LinkedList<Integer>();
    LinkedList<TState> open = new LinkedList<TState>();
    open.add(startState);
    startState.setG(0);
    startState.setH(rules.getH(startState));

    while (!open.isEmpty()) {
        TState x = getStateWithMinF(open);
        if (rules.isTerminate(x)) {
            closedStates = close.size();
            return completeSolution(x);
        }
        open.remove(x);
        close.add(x.hashCode());
        List<TState> neighbors = rules.getNeighbors(x);
        for (TState neighbor : neighbors) {
            if (close.contains(neighbor.hashCode())) {
                continue;
            }
            int g = x.getG() + rules.getDistance(x, neighbor);
            boolean isGBetter;
            if (!open.contains(neighbor)) {
                neighbor.setH(rules.getH(neighbor));
                open.add(neighbor);
                isGBetter = true;
            } else {
                isGBetter = g < neighbor.getG();
            }
            if (isGBetter) {
                neighbor.setParent(x);
                neighbor.setG(g);
            }
        }
    }
    return null;
}

public int getClosedStatesCount() {
    return closedStates;
}

/**
 * Создает объект для поиска терминального состояния по указанным
 * правилам.
 *
 * @param rules правила, в соответствии с которыми будет производиться
 * поиск
 * терминального состояния.
 */
public Astar(TRules rules) {
    if (rules == null) {
        throw new IllegalArgumentException("Rules can't be null.");
    }
    this.rules = rules;
}

/**
 * Находит вершину в списке open с наименьшим значением веса.
 */

```



```

    * @param open список открытых вершин.
    * @return вершину с наименьшим весом.
    */
private TState getStateWithMinF(Collection<TState> open) {
    TState res = null;
    int min = Integer.MAX_VALUE;
    for (TState state : open) {
        if (state.getF() < min) {
            min = state.getF();
            res = state;
        }
    }
    return res;
}

/**
 * Составляет последовательность состояний пройденных от начального
 * состояния до конечного.
 *
 * @param terminate найденное конечное состояние.
 * @return последовательность состояний пройденных от начального
 * состояния до конечного.
 */
private Collection<State> completeSolution(TState terminate) {
    LinkedList<State> path = new LinkedList<State>();
    State c = terminate;
    while (c != null) {
        path.addFirst(c);
        c = c.getParent();
    }
    return path;
}

private TRules rules;
private int closedStates = 0;
}

```

```

package ThirdLab.Algorithm_Astar;

import java.util.List;

/**
 * Определяет специфичные для задачи правила ее решения.
 *
 * @author: dok
 */
public interface Rules<TState extends State> {

    /**
     * Возвращает список состояний, в которые может быть осуществлен переход
из
     * указанного состояния.
     *
     * @param currentState
     *         текущее состояние, для которого раскрываются соседние.
     * @return список состояний, в которые может быть осуществлен переход из
     *         указанного состояния.
     */
    List<TState> getNeighbors(TState currentState);

    /**
     * Возвращает расстояние между указанными состояниями.
     *

```

```

    * @param a
    *         первое состояние.
    * @param b
    *         второе состояние.
    * @return расстояние между указанными состояниями.
    */
    int getDistance(TState a, TState b);

    /**
     * Вычисляет эвристическую оценку расстояния от указанного состояния до
     * конечного.
     *
     * @param state
     *         текущее состояние.
     * @return значение оценки расстояния от указанного состояния до
     * конечного.
     */
    int getH(TState state);

    /**
     * Проверяет состояние, не является ли оно конечным.
     *
     * @param state
     *         состояние.
     * @return true, если состояние конечное.
     */
    boolean isTerminate(TState state);
}

```

```

package ThirdLab.Algorithm_Astar;

/**
 * Представляет вершину графа решений.
 */
public abstract class State {

    /**
     * Возвращает вес состояния как сумму расстояния (от начального состояния
     * до текущего) и эвристической оценки (предполагаемого расстояния от
     * текущего состояния до терминального).
     */
    public int getF() {
        return g + h;
    }

    /**
     * Возвращает расстояние от начального состояния до текущего.
     */
    public int getG() {
        return g;
    }

    /**
     * Устанавливает значение оценки расстояния от начального состояния до
     * текущего.
     */
    public void setG(int g) {
        this.g = g;
    }

    /**
     * Возвращает эвристическую оценку расстояния от текущего состояния до
     * терминального.
     */
}

```

```

    */
    public int getH() {
        return h;
    }

    /**
     * Устанавливает значение эвристической оценки расстояния от текущего
     * состояния до конечного.
     */
    public void setH(int h) {
        this.h = h;
    }

    /**
     * Возвращает предшествующее состояние.
     */
    public State getParent() {
        return parent;
    }

    public void setParent(State parent) {
        this.parent = parent;
    }

    public State(State parent) {
        this.parent = parent;
    }

    private int g;
    private int h;
    private State parent;
}

```

```

package ThirdLab.Fifteen;

import ThirdLab.Algorithm_Astar.Astar;
import ThirdLab.Algorithm_Astar.State;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.util.Collection;
import java.util.Random;

public class FifteenClass {

    public static void main(String[] args) {
        parseArgs(args);

        if (isReadFromStream) {
            try {
                startField = readStartState();
            } catch (IOException e) {
                e.printStackTrace();
                System.exit(1);
            }
            if (sideSize == 4 && !FifteenState.checkState(startField)) {
                System.out
                    .println("\nДанное состояние нельзя привести к
терминальному.\n"
                        + "См.
http://ru.wikipedia.org/wiki/Пятнашки\n");
            }
        }
    }
}

```

```

        System.exit(1);
    }
}

int size = sideSize * sideSize;
terminateField = getTerminalState(sideSize, size);

FifteenRules rules = new FifteenRules2(sideSize, terminateField);
FifteenState startState = new FifteenState(null, sideSize);

if (startField == null) {
    startField = generateStartState(rules, stepCount);
}
startState.setField(startField);

Astar<FifteenState, FifteenRules> astar = new Astar<FifteenState,
FifteenRules>(
    rules);
long time = System.currentTimeMillis();
Collection<State> res = astar.search(startState);
time = System.currentTimeMillis() - time;

if (res == null) {
    System.out.println("Решение не найдено.");
    return;
} else {
    for (State s : res) {
        System.out.println(s.toString());
    }
}
if (isShowStatistic) {
    System.out.println("Время: " + time + "мс");
    /* Начальное состояние за ход не считается */
    System.out.println("Длина решения: " + (res.size() - 1));
    System.out
        .println("Открытые состояния: " +
astar.getClosedStatesCount());
}
}

/**
 * Считывает начальное состояние из входного потока, определяя
размерность
 * поля по количеству строк в потоке.
 *
 * @return массив байт, описывающий начальное состояние или null, если не
удалось прочесть начальное состояние.
 * @throws IOException
 */
private static byte[] readStartState() throws IOException {
    System.out.println("Reading state from input stream...");
    InputStreamReader istr = new InputStreamReader(System.in);
    BufferedReader reader = new BufferedReader(istr);
    String line = null;
    sideSize = 0;
    StringBuffer buf = new StringBuffer();
    while ((line = reader.readLine()) != null) {
        if (line.isEmpty()) {
            break;
        }
        buf.append(line + "\n");
        sideSize++;
    }
    String state = buf.toString();

```

```

        if (state.isEmpty()) {
            return null;
        } else {
            return FifteenState.parseField(state);
        }
    }

    /**
     * Генерирует начальное состояние путем swapCount начальных перестановок.
     *
     * @param rules
     * @param swapCount
     *             количество перестановок.
     * @return сгенерированное начальное состояние.
     */
    private static byte[] generateStartState(FifteenRules rules, int
swapCount) {
        int stepCount = swapCount;
        byte[] startState = rules.getTerminateState();

        int[] actions = rules.getActions();
        Random r = new Random();
        while (stepCount > 0) {
            int j = r.nextInt(actions.length);
            byte[] state = rules.doAction(startState, actions[j]);
            if (state != null) {
                startState = state;
                stepCount--;
            }
        }
        return startState;
    }

    /**
     * Генерирует терминальное состояние, как упорядоченную
последовательность
     * чисел.
     *
     * @param sideSize
     * @param size
     * @return
     */
    private static byte[] getTerminalState(int sideSize, int size) {
        if (terminateField == null) {
            terminateField = new byte[size];
            byte k = 0;
            for (int i = 0; i < sideSize; i++) {
                for (int j = 0; j < sideSize; j++) {
                    terminateField[j + i * sideSize] = ++k;
                }
            }
            terminateField[size - 1] = 0;
        }
        return terminateField;
    }

    /**
     * Разбирает аргументы запуска приложения.
     *
     * @param args
     */
    private static void parseArgs(String[] args) {
        if (args == null || args.length == 0) {
            return;
        }
    }

```

```

    }
    for (int i = 0; i < args.length; i++) {
        if (args[i].equals("-h")) {
            try {
                showHelp();
            } catch (IOException e) {
                e.printStackTrace();
            }
            continue;
        }
        if (args[i].equals("-v")) {
            isShowStatistic = true;
            continue;
        }
        if (args[i].equals("-s")) {
            isReadFromStream = false;
            sideSize = Integer.parseInt(args[++i]);
            continue;
        }
        if (args[i].equals("-c")) {
            isReadFromStream = false;
            stepCount = Integer.parseInt(args[++i]);
            continue;
        }
        throw new IllegalArgumentException("Unknown argument: " +
args[i]);
    }
}

private static void showHelp() throws IOException {
    InputStreamReader strm = new InputStreamReader(
        FifteenClass.class.getResourceAsStream("/help.ru"), "UTF-8");
    BufferedReader reader = new BufferedReader(strm);

    PrintStream out = new PrintStream(System.out, true);

    String str = null;
    while ((str = reader.readLine()) != null) {
        out.println(str);
    }
    reader.close();
    System.exit(0);
}

private static byte[] startField;           // начальное поле
private static byte[] terminateField;       // конечное поле
private static int stepCount = 10;          // счётчик шагов
private static int sideSize = 4;            // Размер стороны

private static boolean isReadFromStream = false; // Чтение из потока

private static boolean isShowStatistic = true; // Показать статистику
(Время, длину решения, кол-во открытых состояний)
}

```

```

package ThirdLab.Fifteen;

import ThirdLab.Algorithm_Astar.Rules;
import ThirdLab.Algorithm_Astar.State;

```

```

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * Определяет специфичные для задачи правила ее решения. В данной реализации
 * эвристика вычисляется как количество клеток, находящихся не на своих
 * местах.
 */
public class FifteenRules implements Rules<FifteenState> {

    public List<FifteenState> getNeighbors(FifteenState currentState) {
        ArrayList<FifteenState> res = new ArrayList<FifteenState>();
        for (int i = 0; i < actions.length; i++) {
            byte[] field = doAction(currentState.getField(), actions[i]);
            if (field == null) {
                continue;
            }
            FifteenState state = new FifteenState(currentState, sideSize);
            state.setField(field);
            res.add(state);
        }
        return res;
    }

    /**
     * Подсчитывает количество родительских состояний от a до b.
     *
     * @param a
     *         первое состояние. Должно быть среди состояний,
     *         предшествующих
     *         b.
     * @param b
     *         второе состояние.
     * @return количество переходов от a до b.
     */
    public int getDistance(FifteenState a, FifteenState b) {
        State c = b;
        int res = 0;
        while ((c != null) && (!c.equals(a))) {
            c = c.getParent();
            res++;
        }
        return res;
    }

    /**
     * На самом деле, в силу специфики реализации A*, данному методу
     * достаточно всегда возвращать 1.
     */

    /**
     * Эвристика вычисляется как количество клеток, находящихся не на своих
     * местах.
     */
    public int getH(FifteenState state) {
        int res = 0;
        for (int i = 0; i < size; i++) {
            if (state.getField()[i] != terminateState[i]) {
                res++;
            }
        }
        return res;
    }
}

```

```

public boolean isTerminate(FifteenState state) {
    return Arrays.equals(state.getField(), terminateState);
}

public byte[] getTerminateState() {
    return terminateState;
}

/**
 * Возвращает массив доступных действий.
 */
public int[] getActions() {
    return actions;
}

/**
 * Применяет к состоянию правило.
 *
 * @param field начальное состояние.
 * @param action применяемое правило.
 * @return новое состояние, полученное в результате применения правила.
null
 * если состояние недопустимо.
 */
public byte[] doAction(byte[] field, int action) {
    /* Выполняется поиск пустой клетки */
    int zero = 0;
    for (; zero < field.length; zero++) {
        if (field[zero] == 0) {
            break;
        }
        if (zero >= field.length) {
            return null;
        }
    }
    /* Вычисляется индекс перемещаемой клетки */
    int number = zero + action;
    /* Проверяется допустимость хода */
    if (number < 0 || number >= field.length) {
        return null;
    }
    if ((action == 1) && ((zero + 1) % sideSize == 0)) {
        return null;
    }
    if ((action == -1) && ((zero + 1) % sideSize == 1)) {
        return null;
    }
    /*
    * Создается новый экземпляр поля, на котором меняются местами пустая
и
    * перемещаемая клетки
    */
    byte[] newField = Arrays.copyOf(field, field.length);
    byte temp = newField[zero];
    newField[zero] = newField[number];
    newField[number] = temp;

    return newField;
}

/**
 * @param fieldSize

```



```

        *           размер поля (количество клеток на одной стороне).
        * @param terminateState
        *           конечное состояние.
        */
    public FifteenRules(int fieldSize, byte[] terminateState) {
        if (fieldSize < 2) {
            throw new IllegalArgumentException("Invalid field size.");
        }
        if (terminateState == null) {
            throw new IllegalArgumentException("Terminate state can't be
null.");
        }

        this.sideSize = fieldSize;
        size = sideSize * sideSize;

        if (terminateState.length != size) {
            throw new IllegalArgumentException(
                "Size of terminate state is incorrect.");
        }
        this.terminateState = terminateState;

        top = -sideSize;
        bottom = sideSize;

        actions = new int[] { top, bottom, left, right };
    }

    protected int sideSize;
    protected int size;

    protected byte[] terminateState;

    private int left = -1;
    private int top;
    private int right = 1;
    private int bottom;
    protected int[] actions;
}

```

```

package ThirdLab.Fifteen;

/**
 * User: pva
 * Date: 13.03.12
 * Time: 10:46
 */
public class FifteenRules2 extends FifteenRules {

    /** Эвристика: нарушение порядка на первых строках штрафуются сильнее. */
    @Override
    public int getH(FifteenState state) {
        int res = 0;
        int penalty = sideSize;
        for (int i = 0; i < size; i++) {
            if ((i+1) % sideSize == 0) {
                penalty--;
            }
            if (state.getField()[i] != terminateState[i]) {
                res += penalty;
            }
        }
        return res;
    }
}

```

```

    }

    /**
     * @param fieldSize      размер поля (количество клеток на одной
    стороне).
     * @param terminateState конечное состояние.
     */
    public FifteenRules2(int fieldSize, byte[] terminateState) {
        super(fieldSize, terminateState);
    }
}

```

```

package ThirdLab.Fifteen;

import ThirdLab.Algorithm_Astar.State;

import java.util.Arrays;

/**
 * Представляет состояние игрового поля головоломки "Пятнашки".
 */
public class FifteenState extends State {

    public static byte[] parseField(String str) {
        int i = 0;
        String[] lines = str.split("\n");
        byte[] res = new byte[lines.length * lines.length];
        for (String line : lines) {
            String[] vals = line.trim().replaceAll("\\s+", ":").split(":");
            for (String v : vals) {
                res[i] = Byte.parseByte(v.trim());
                i++;
            }
        }
        return res;
    }

    /**
     * Проверяет, возможно ли привести состояние к терминальному.
     *
     * @param field
     *      состояние игрового поля.
     * @return true - если можно привести к терминальному.
     *
     * @see <a
    href="https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%B0_%D0%B2_15">Wikip
    edia: Игра_в_15</a>
     */
    public static boolean checkState(byte[] field) {
        int N = 0;
        int e = 0;
        int sideSize = 4;
        for (int i = 0; i < field.length; i++) {
            /* Определяется номер ряда пустой клетки (считая с 1). */
            if (field[i] == 0) {
                e = i / sideSize + 1;
            }
            if (i == 0)
                continue;
            /* Производится подсчет количества клеток меньших текущей */
            for (int j = i + 1; j < field.length; j++) {
                if (field[j] < field[i]) {
                    N++;
                }
            }
        }
    }
}

```

```

    }
}

N = N + e;
/* Если N является нечётной, то решения головоломки не существует. */
return (N & 1) == 0; // Первый бит четного числа равен 0
}

/**
 * Возвращает состояние игрового поля в виде одномерного массива байт.
 */
public byte[] getField() {
    return field;
}

/**
 * Устанавливает состояние игрового поля.
 */
public void setField(byte[] field) {
    this.field = field;
    hash = Arrays.hashCode(field);
}

@Override
public String toString() {
    if (field == null) {
        return "" + null;
    }
    StringBuffer sbf;
    sbf = new StringBuffer(field.length);
    for (int i = 0; i < sideSize; i++) {
        for (int j = 0; j < sideSize; j++) {
            sbf.append(field[j + i * sideSize]);
            sbf.append("\t");
        }
        sbf.append("\n");
    }
    return sbf.toString();
}

@Override
public boolean equals(Object obj) {
    if (obj == null || !(obj instanceof FifteenState)) {
        return false;
    }
    return hash == obj.hashCode();
}

@Override
public int hashCode() {
    return hash;
}

/**
 * Создает описание состояния игрового поля.
 *
 * @param parent
 *     предшествующее состояние.
 * @param sideSize
 *     размер стороны поля.
 */
public FifteenState(State parent, int sideSize) {
    super(parent);
    this.sideSize = sideSize;
}

```

```
}  
  
private byte[] field;  
private int sideSize;  
private int hash;  
}
```

3. Вывод

Мы реализовали метод генерации массива, а также алгоритмы указанных в условии сортировок.