Федеральное агентство связи

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Московский технический университет связи и информатики»

Кафедра «Информатики»

Дисциплина «СиАОД»

Лабораторная работа №3

Выполнил: студент группы БВТ1901

Адедиха Коффи Жермен

Руководитель:

Мелехин А.

Москва 2021

Лабораторная работа 3.

Методы поиска подстроки в строке.

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

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.

ВЫПОЛНЕНИЕ РАЬОТЫ

Задание 1

- **Кнута-Морриса-Пратта**

```java
import java.util.*;

class KMP_String_Matching {

    void KMPSearch(String pat, String txt)
    {
        int M = pat.length();
        int N = txt.length();

        // create lps[] that will hold the longest
        // prefix suffix values for pattern
        int lps[] = new int[M];
        int j = 0; // index for pat[]

        // Preprocess the pattern (calculate lps[]
        // array)
        computeLPSArray(pat, M, lps);

        int i = 0; // index for txt[]
        while (i < N) {
            if (pat.charAt(j) == txt.charAt(i)) {
                j++;
                i++;
            }
            if (j == M) {
                System.out.println("Found pattern "
                                   + "at index " + (i - j));
                j = lps[j - 1];
            }

            // mismatch after j matches
            else if (i < N && pat.charAt(j) != txt.charAt(i)) {
                // Do not match lps[0..lps[j-1]] characters,
                // they will match anyway
                if (j != 0)
                    j = lps[j - 1];
                else
                    i = i + 1;
```

```
            }

        }

    }

    void computeLPSArray(String pat, int M, int lps[])
    {
        // length of the previous longest prefix suffix
        int len = 0;
        int i = 1;
        lps[0] = 0; // lps[0] is always 0

        // the loop calculates lps[i] for i = 1 to M-1
        while (i < M) {
            if (pat.charAt(i) == pat.charAt(len)) {
                len++;
                lps[i] = len;
                i++;
            }
            else
            {

                if (len != 0) {
                    len = lps[len - 1];

                    // Also, note that we do not increment
                    // i here
                }
                else // if (len == 0)
                {
                    lps[i] = len;
                    i++;
                }
            }
        }
    }

    // Driver program to test above function
    public static void main(String args[])
    {

        String txt = "Germain koffi Adediha";
        String pat = "koffi";
```

```
        new KMP_String_Matching().KMPSearch(pat, txt);

    }
}
```

- **Упрощенный Бойера-Мура**

```
import java.util.*;
public class BoyerMoore{

public static int getFirstEntry(String source, String template) {
    int sourceLen = source.length();
    int templateLen = template.length();
    if (templateLen > sourceLen) {
        return -1;
    }
    HashMap<Character, Integer> offsetTable = new HashMap<Character, Integer>();
    for (int i = 0; i <= 255; i++) {
        offsetTable.put((char) i, templateLen);
    }
    for (int i = 0; i < templateLen - 1; i++) {
        offsetTable.put(template.charAt(i), templateLen - i - 1);
    }
    int i = templateLen - 1;
    int j = i;
    int k = i;
    while (j >= 0 && i <= sourceLen - 1) {
        j = templateLen - 1;
        k = i;
        while (j >= 0 && source.charAt(k) == template.charAt(j)) {
            k--;
            j--;
        }
        i += offsetTable.get(source.charAt(i));
    }
    if (k >= sourceLen - templateLen) {
        return -1;
    } else {
        return k + 1;
```

```
    }
}
public static void main(String[] args) {
    String str="Germain";
    String Str2="KOffiGermain";
    System.out.print("Pattern found at "+ getFirstEntry(Str2,str));
}
}
```

Задание 2 : Пятнашки

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Queue;


public class FifteenPuzzle {

  private class TilePos {
        public int x;
        public int y;

        public TilePos(int x, int y) {
            this.x=x;
            this.y=y;
        }

    }

    public final static int DIMS=4;
    private int[][] tiles;
    private int display_width;
    private TilePos blank;

    public FifteenPuzzle() {
        tiles = new int[DIMS][DIMS];
        int cnt=1;
```

```
        for(int i=0; i<DIMS; i++) {
            for(int j=0; j<DIMS; j++) {
                tiles[i][j]=cnt;
                cnt++;
            }
        }
        display_width=Integer.toString(cnt).length();

        // init blank
        blank = new TilePos(DIMS-1,DIMS-1);
        tiles[blank.x][blank.y]=0;
    }

    public final static FifteenPuzzle SOLVED=new FifteenPuzzle();


    public FifteenPuzzle(FifteenPuzzle toClone) {
        this();  // chain to basic init
        for(TilePos p: allTilePos()) {
            tiles[p.x][p.y] = toClone.tile(p);
        }
        blank = toClone.getBlank();
    }

    public List<TilePos> allTilePos() {
        ArrayList<TilePos> out = new ArrayList<TilePos>();
        for(int i=0; i<DIMS; i++) {
            for(int j=0; j<DIMS; j++) {
                out.add(new TilePos(i,j));
            }
        }
        return out;
    }


    public int tile(TilePos p) {
        return tiles[p.x][p.y];
    }


    public TilePos getBlank() {
        return blank;
    }
```

```
public TilePos whereIs(int x) {
    for(TilePos p: allTilePos()) {
        if( tile(p) == x ) {
            return p;
        }
    }
    return null;
}


@Override
public boolean equals(Object o) {
    if(o instanceof FifteenPuzzle) {
        for(TilePos p: allTilePos()) {
            if( this.tile(p) != ((FifteenPuzzle) o).tile(p)) {
                return false;
            }
        }
        return true;
    }
    return false;
}


@Override
public int hashCode() {
    int out=0;
    for(TilePos p: allTilePos()) {
        out= (out*DIMS*DIMS) + this.tile(p);
    }
    return out;
}


public void show() {
    System.out.println("-----------------");
    for(int i=0; i<DIMS; i++) {
        System.out.print("| ");
        for(int j=0; j<DIMS; j++) {
            int n = tiles[i][j];
            String s;
            if( n>0) {
                s = Integer.toString(n);
```

```
            } else {
                s = "";
            }
            while( s.length() < display_width ) {
                s += " ";
            }
            System.out.print(s + "| ");
        }
        System.out.print("\n");
    }
    System.out.print("-----------------\n\n");
}


public List<TilePos> allValidMoves() {
    ArrayList<TilePos> out = new ArrayList<TilePos>();
    for(int dx=-1; dx<2; dx++) {
        for(int dy=-1; dy<2; dy++) {
            TilePos tp = new TilePos(blank.x + dx, blank.y + dy);
            if( isValidMove(tp) ) {
                out.add(tp);
            }
        }
    }
    return out;
}


public boolean isValidMove(TilePos p) {
    if( ( p.x < 0) || (p.x >= DIMS) ) {
        return false;
    }
    if( ( p.y < 0) || (p.y >= DIMS) ) {
        return false;
    }
    int dx = blank.x - p.x;
    int dy = blank.y - p.y;
    if( (Math.abs(dx) + Math.abs(dy) != 1 ) || (dx*dy != 0) ) {
        return false;
    }
    return true;
}
```

```
    public void move(TilePos p) {
        if( !isValidMove(p) ) {
            throw new RuntimeException("Invalid move");
        }
        assert tiles[blank.x][blank.y]==0;
        tiles[blank.x][blank.y] = tiles[p.x][p.y];
        tiles[p.x][p.y]=0;
        blank = p;
    }



    public FifteenPuzzle moveClone(TilePos p) {
        FifteenPuzzle out = new FifteenPuzzle(this);
        out.move(p);
        return out;
    }


    public void shuffle(int howmany) {
        for(int i=0; i<howmany; i++) {
            List<TilePos> possible = allValidMoves();
            int which =  (int) (Math.random() * possible.size());
            TilePos move = possible.get(which);
            this.move(move);
        }
    }


    public void shuffle() {
        shuffle(DIMS*DIMS*DIMS*DIMS*DIMS);
    }


    public int numberMisplacedTiles() {
        int wrong=0;
        for(int i=0; i<DIMS; i++) {
            for(int j=0; j<DIMS; j++) {
                if( (tiles[i][j] >0) && ( tiles[i][j] != SOLVED.tiles[i][j] ) ){
                    wrong++;
                }
            }
        }
        return wrong;
```

```
    }


    public boolean isSolved() {
        return numberMisplacedTiles() == 0;
    }



    public int manhattanDistance() {
        int sum=0;
        for(TilePos p: allTilePos()) {
            int val = tile(p);
            if( val > 0 ) {
                TilePos correct = SOLVED.whereIs(val);
                sum += Math.abs( correct.x = p.x );
                sum += Math.abs( correct.y = p.y );
            }
        }
        return sum;
    }


    public int estimateError() {
        return this.numberMisplacedTiles();
        //return 5*this.numberMisplacedTiles(); // finds a non-
optimal solution faster
        //return this.manhattanDistance();
    }


    public List<FifteenPuzzle> allAdjacentPuzzles() {
        ArrayList<FifteenPuzzle> out = new ArrayList<FifteenPuzzle>();
        for( TilePos move: allValidMoves() ) {
            out.add( moveClone(move) );
        }
        return out;
    }

    public List<FifteenPuzzle> dijkstraSolve() {
        Queue<FifteenPuzzle> toVisit = new LinkedList<FifteenPuzzle>();
        HashMap<FifteenPuzzle,FifteenPuzzle> predecessor = new HashMap<FifteenPuz
zle,FifteenPuzzle>();
        toVisit.add(this);
```

```
        predecessor.put(this, null);
        int cnt=0;
        while( toVisit.size() > 0) {
            FifteenPuzzle candidate = toVisit.remove();
            cnt++;
            if( cnt % 10000 == 0) {
                System.out.printf("Considered %,d positions. Queue = %,d\n", cnt,
 toVisit.size());
            }
            if( candidate.isSolved() ) {
                System.out.printf("Solution considered %d boards\n", cnt);
                LinkedList<FifteenPuzzle> solution = new LinkedList<FifteenPuzzle
>();
                FifteenPuzzle backtrace=candidate;
                while( backtrace != null ) {
                    solution.addFirst(backtrace);
                    backtrace = predecessor.get(backtrace);
                }
                return solution;
            }
            for(FifteenPuzzle fp: candidate.allAdjacentPuzzles()) {
                if( !predecessor.containsKey(fp) ) {
                    predecessor.put(fp,candidate);
                    toVisit.add(fp);
                }
            }
        }
        return null;
    }

    public List<FifteenPuzzle> aStarSolve() {
        HashMap<FifteenPuzzle,FifteenPuzzle> predecessor = new HashMap<FifteenPuz
zle,FifteenPuzzle>();
        HashMap<FifteenPuzzle,Integer> depth = new HashMap<FifteenPuzzle,Integer>
();
        final HashMap<FifteenPuzzle,Integer> score = new HashMap<FifteenPuzzle,In
teger>();
        Comparator<FifteenPuzzle> comparator = new Comparator<FifteenPuzzle>() {
            @Override
            public int compare(FifteenPuzzle a, FifteenPuzzle b) {
                return score.get(a) - score.get(b);
            }
        };
```

```
        PriorityQueue<FifteenPuzzle> toVisit = new PriorityQueue<FifteenPuzzle>(1
0000,comparator);

        predecessor.put(this, null);
        depth.put(this,0);
        score.put(this, this.estimateError());
        toVisit.add(this);
        int cnt=0;
        while( toVisit.size() > 0) {
            FifteenPuzzle candidate = toVisit.remove();
            cnt++;
            if( cnt % 10000 == 0) {
                System.out.printf("Considered %,d positions. Queue = %,d\n", cnt,
 toVisit.size());
            }
            if( candidate.isSolved() ) {
                System.out.printf("Solution considered %d boards\n", cnt);
                LinkedList<FifteenPuzzle> solution = new LinkedList<FifteenPuzzle
>();
                FifteenPuzzle backtrace=candidate;
                while( backtrace != null ) {
                    solution.addFirst(backtrace);
                    backtrace = predecessor.get(backtrace);
                }
                return solution;
            }
            for(FifteenPuzzle fp: candidate.allAdjacentPuzzles()) {
                if( !predecessor.containsKey(fp) ) {
                    predecessor.put(fp,candidate);
                    depth.put(fp, depth.get(candidate)+1);
                    int estimate = fp.estimateError();
                    score.put(fp, depth.get(candidate)+1 + estimate);
                    // dont' add to p-
queue until the metadata is in place that the comparator needs
                    toVisit.add(fp);
                }
            }
        }
        return null;
    }

    private static void showSolution(List<FifteenPuzzle> solution) {
        if (solution != null ) {
```

```
            System.out.printf("Success!  Solution with %d moves:\n", solution.siz
e());
            for( FifteenPuzzle sp: solution) {
                sp.show();
            }
        } else {
            System.out.println("Did not solve. :(");
        }
    }


    public static void main(String[] args) {
        FifteenPuzzle p = new FifteenPuzzle();
        p.shuffle(70);  // Number of shuffles is critical -
- large numbers (100+) and 4x4 puzzle is hard even for A*.
        System.out.println("Shuffled board:");
        p.show();

        List<FifteenPuzzle> solution;

        System.out.println("Solving with A*");
        solution = p.aStarSolve();
        showSolution(solution);


        solution = p.dijkstraSolve();
        showSolution(solution);
    }

}
```

Работы программы

Задание 1

КМР

```
String txt = "Germain koffi Adediha";
String pat = "koffi";
```

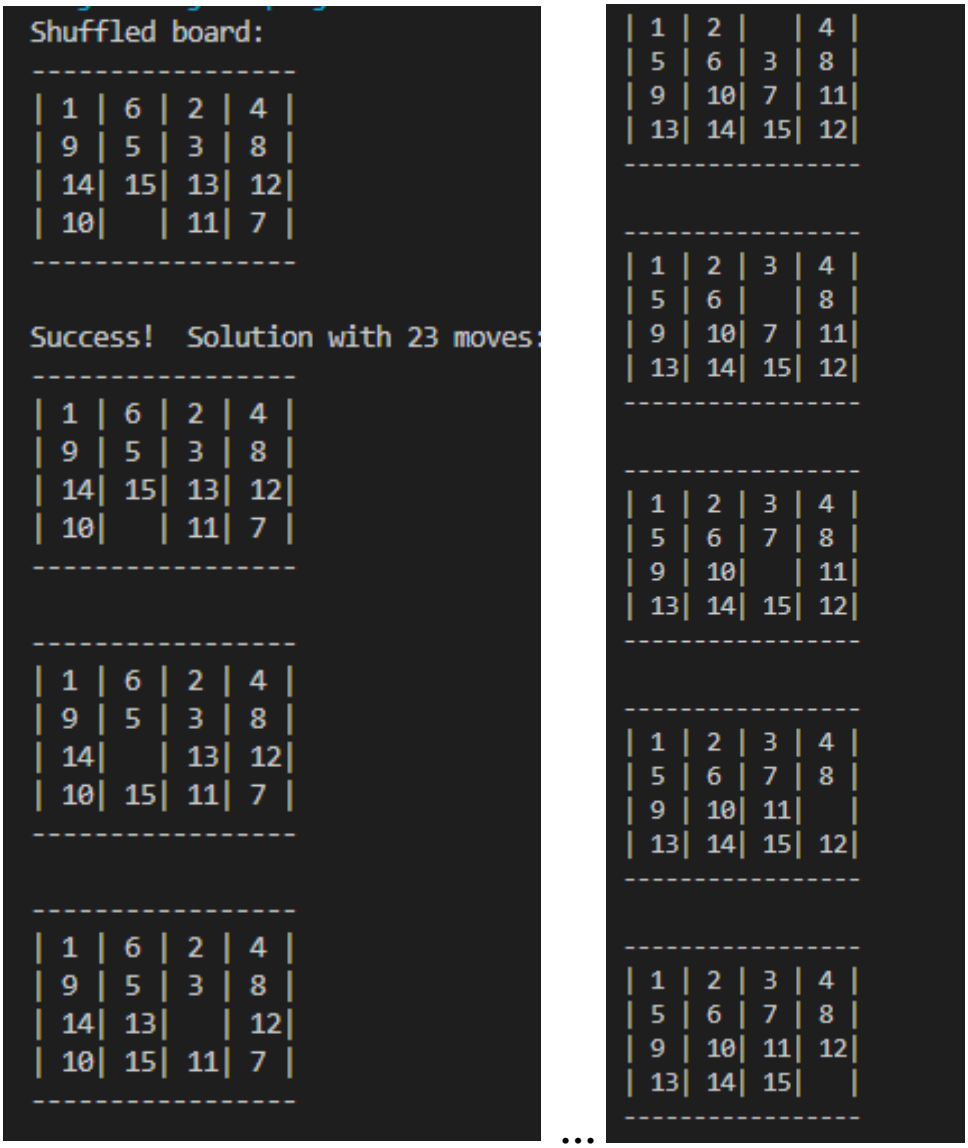
```
Found pattern at index 8
```

Упрощенный Бойера-Мура

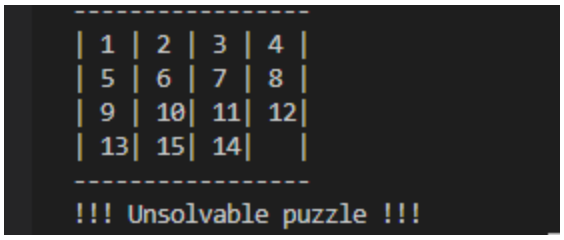
```
String str="Germain";
String Str2="KOffiGermain";
```

```
Pattern found at 5
```

Задание 2 : Пятнашки



...



Неразрешимый случай, найденный нашей программой.

Вывод

С этой работы мы реализовали методы поиска подстроки в строке. Добавили возможность ввода строки и подстроки с клавиатуры. Предусмотрели возможность существования пробела. Мы работали с алгоритмами КМР и Бойера-Мура, и знали, как они работают.