Федеральное агентство связи

Государственное бюджетное образовательное учреждение высшего образование

Ордена Трудового Красного Знамени

«Московский технический университет связи и информатики»

Кафедра «МКиИТ»

дисциплина «СиАОД»

Отчет по Лабораторной работе №3

Подготовила студентка

группы БВТ1901: Нкурикийе Хафидати

Проверил: 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

1.

```
import java.util.Scanner;

public class Find
{
    public static void StringSearch(String Line, String inLine) {
```

```
if (inLine == null || inLine.length() == 0)
{
   System.out.println("Начинается с нулевого индекса");
   return;
}

if (Line == null || inLine.length() > Line.length())
{
   System.out.println("Подстрока не найдена");
   return;
}

char[] chars = inLine.toCharArray();


int[] next = new int[inLine.length() + 1];
for (int i = 1; i < inLine.length(); i++)
{
   int j = next[i + 1];

   while (j > 0 && chars[j] != chars[i])
      j = next[j];

   if (j > 0 || chars[j] == chars[i])
```

```
            next[i + 1] = j + 1;
        }

        for (int i = 0, j = 0; i < Line.length(); i++)
        {
            if (j < inLine.length() && Line.charAt(i) == inLine.charAt(j))
            {
                if (++j == inLine.length())
                {
                    System.out.println("Начинается с индекса: " + (i - j + 1));
                }
            }
            else if (j > 0)
            {
                j = next[j];
                i--;
            }
        }
    }


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

```
Scanner in = new Scanner(System.in);

System.out.print("Введите строку: ");
String text = in.nextLine();
System.out.println();

System.out.print("Добавьте строку: ");
String newText = in.nextLine();
text = text + newText;
System.out.println();
System.out.println("Новая строка: " + text);
System.out.println();

System.out.print("Введите подстроку для поиска: ");
String pattern = in.nextLine();
System.out.println();



System.out.println("Поиск КМП:");
String finalText = text;
long before = System.nanoTime();
StringSearch(finalText, pattern);
long after = System.nanoTime();
System.out.println("Time KMP in nano: " + (after - before));
```

```
        System.out.println();

        System.out.println("Стандартный поиск:");
        System.out.println("Найдено на индексе: " + text.indexOf(pattern));
        String finalText1 = text;
        before = System.nanoTime();
        finalText1.indexOf(pattern);
        after = System.nanoTime();
        System.out.println("Time indexOF in nano: " + (after - before));


    }
}
```

2.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
```

```
/** Class BoyerMoore **/
public class BoyerMoore
{
    /** function 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);
    }

    /** Function to calculate index of pattern substring **/
    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, j; i < text.length;)
        {
```

```
        for (j = pattern.length - 1; pattern[j] == text[i]; --i, --j)
            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;
}



/** Makes the jump table based on the mismatched character information **/
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;
}
/** Makes the jump table based on the scan offset which mismatch occurs. **/
```

```
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;

}
/** function to check if needle[p:end] a prefix of pattern **/
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;
    }
    /** function to returns the maximum length of the substring ends at p and is a suffix **/
    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
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Boyer Moore Algorithm Test\n");
        System.out.println("\nEnter Text\n");
        String text = br.readLine();
        System.out.println("\nEnter Pattern\n");
        String pattern = br.readLine();
        BoyerMoore bm = new BoyerMoore();
        bm.findPattern(text, pattern);
    }
}
```

ЗАДАНИЯ№2

```
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;
}


/**
 * returns a new puzzle with the move applied
 * @param p
 * @return
 */
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;
  }



  /**
   * another A* heuristic.
   * Total manhattan distance (L1 norm) from each non-blank tile to its correct position
   * @return
   */
  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;
  }


  /**
```

```
     * distance heuristic for A*
     * @return
     */
    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;
    }

    /**
     * returns a list of boards if it was able to solve it, or else null
     * @return
     */
    public List<FifteenPuzzle> dijkstraSolve() {
        Queue<FifteenPuzzle> toVisit = new LinkedList<FifteenPuzzle>();
```

```
    HashMap<FifteenPuzzle,FifteenPuzzle> predecessor = new
HashMap<FifteenPuzzle,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);
```

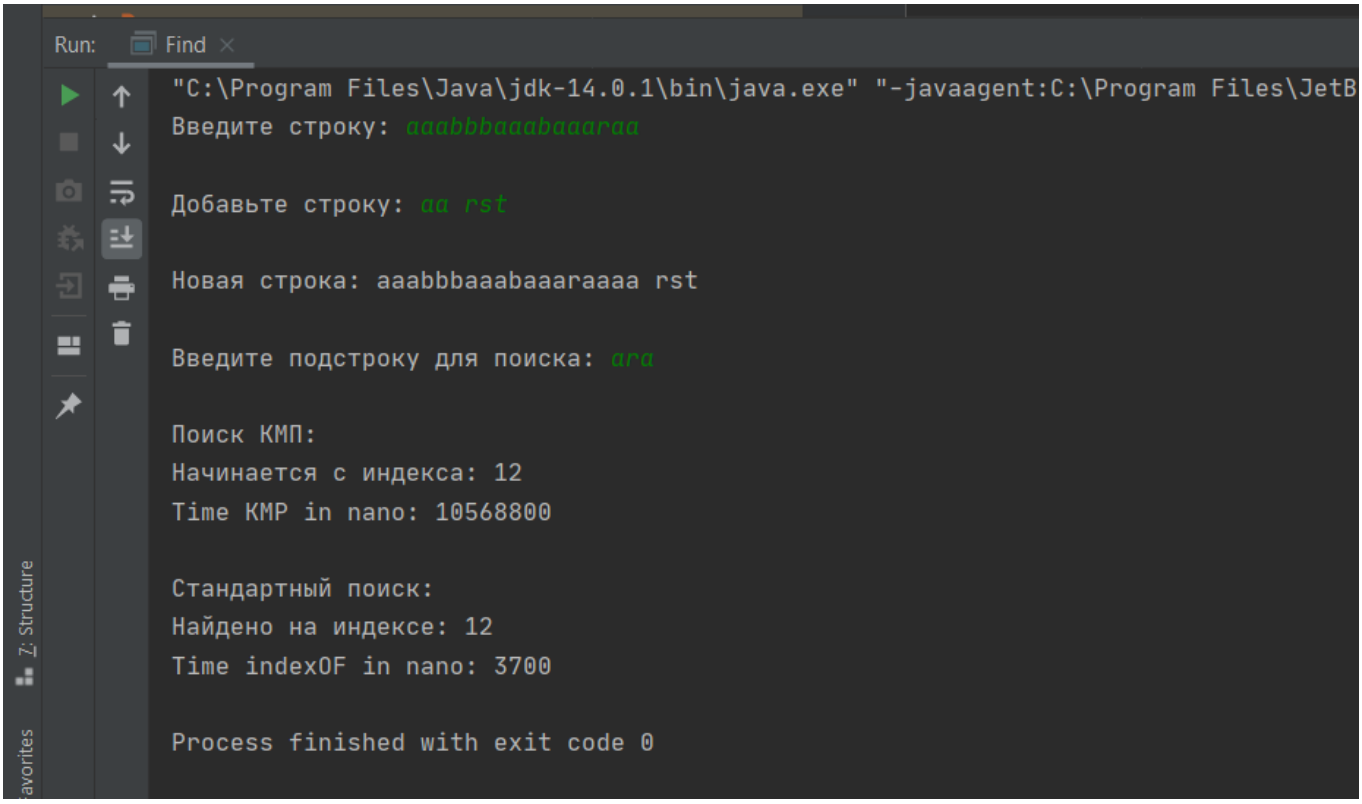
```
                toVisit.add(fp);
            }
        }
    }
    return null;
  }



  /**
   * returns a list of boards if it was able to solve it, or else null
   * @return
   */
  public List<FifteenPuzzle> aStarSolve() {
    HashMap<FifteenPuzzle,FifteenPuzzle> predecessor = new
HashMap<FifteenPuzzle,FifteenPuzzle>();
    HashMap<FifteenPuzzle,Integer> depth = new
HashMap<FifteenPuzzle,Integer>();
    final HashMap<FifteenPuzzle,Integer> score = new
HashMap<FifteenPuzzle,Integer>();
    Comparator<FifteenPuzzle> comparator = new Comparator<FifteenPuzzle>()
{
      @Override
      public int compare(FifteenPuzzle a, FifteenPuzzle b) {
        return score.get(a) - score.get(b);
      }
```

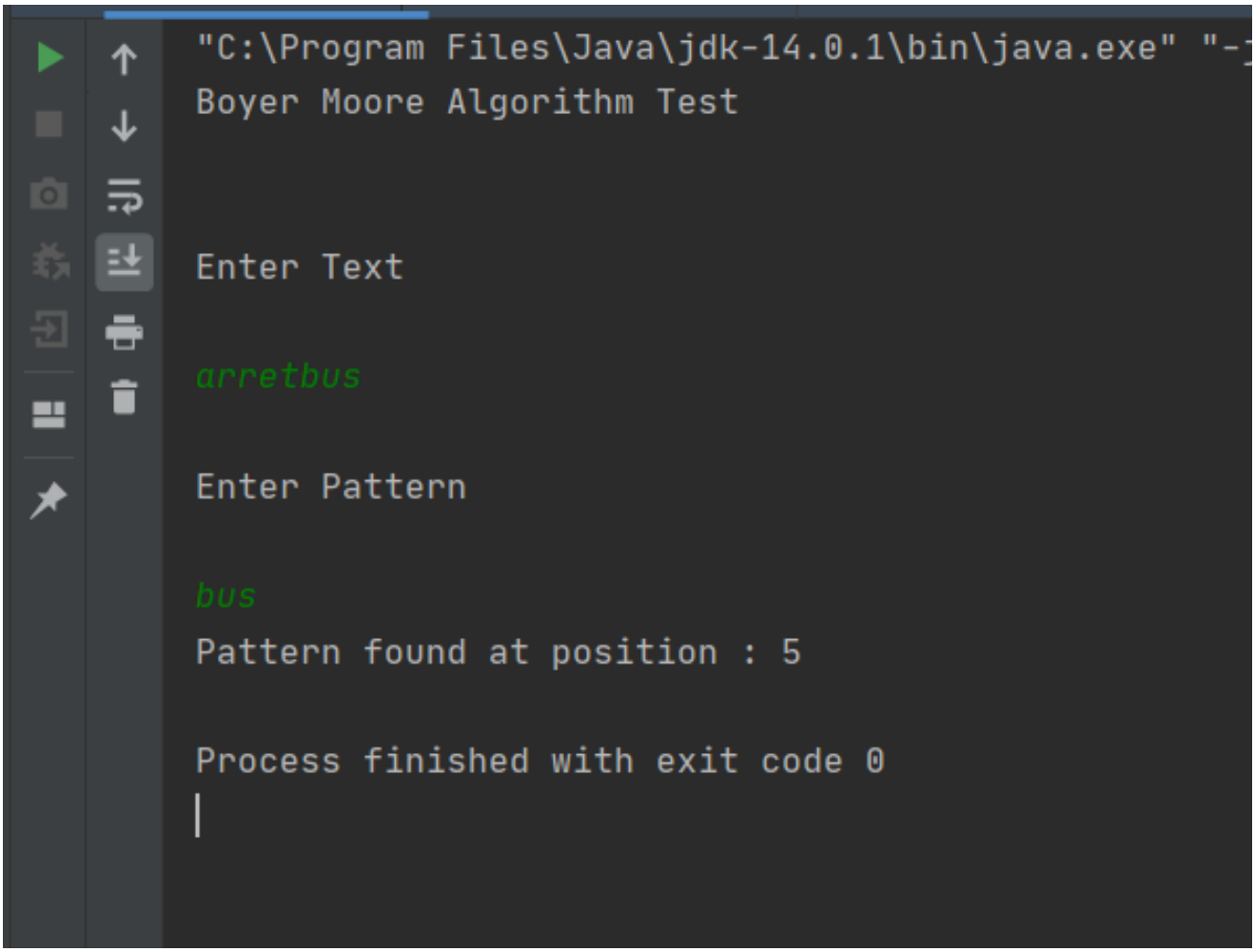
```
    };
    PriorityQueue<FifteenPuzzle> toVisit = new
PriorityQueue<FifteenPuzzle>(10000,comparator);


    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.size());
        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);


        System.out.println("Solving with Dijkstra");
        solution = p.dijkstraSolve();
        showSolution(solution);
    }


}
```

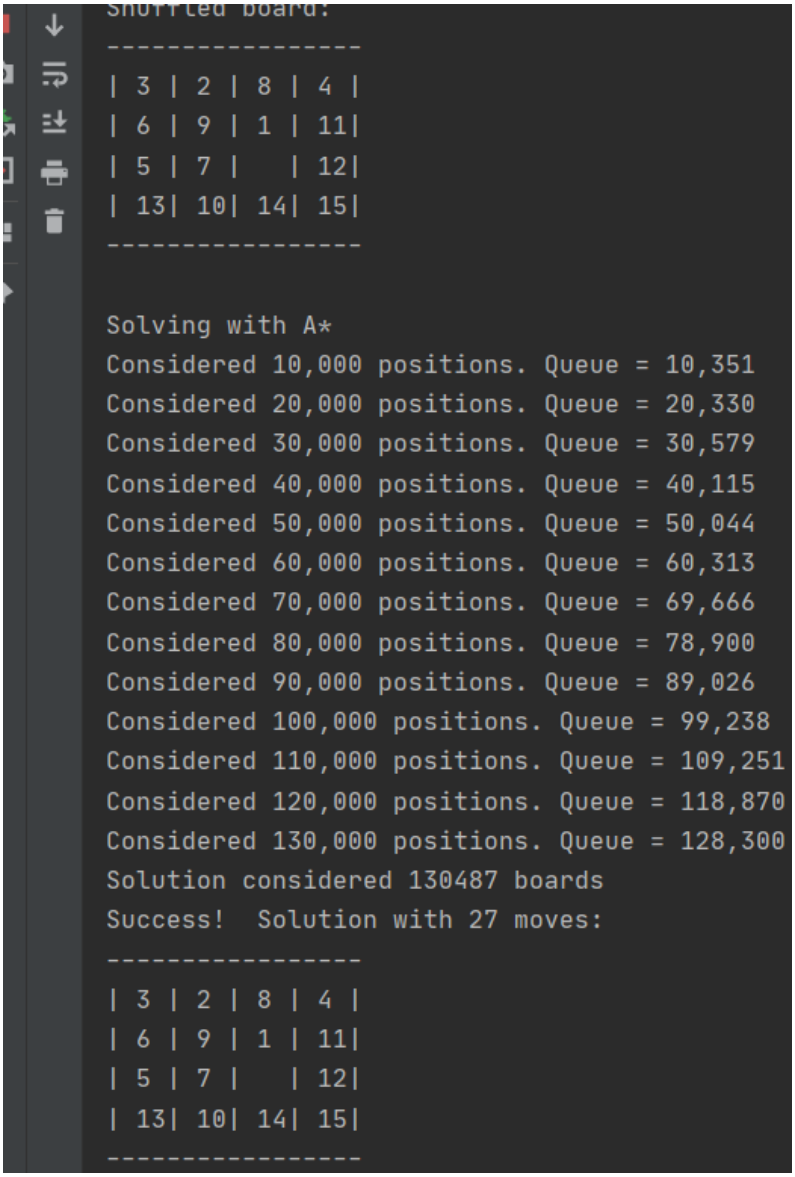
РЕЗУЛЬТАТ

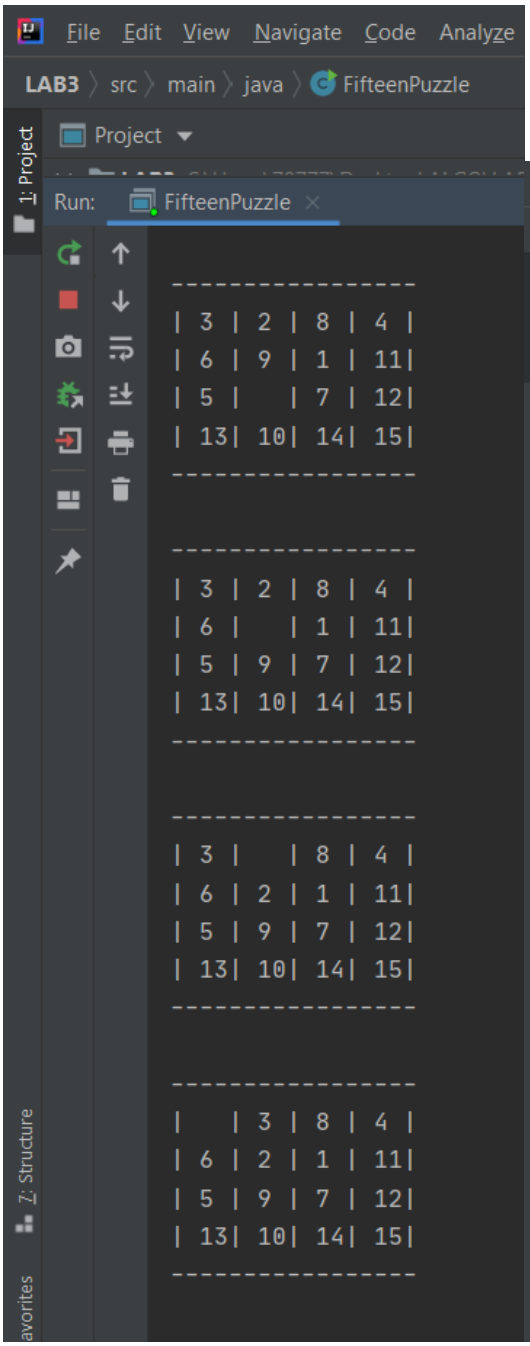

Run:
Find
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetB
Введите строку: aaabbbaaabaaaraa
Добавьте строку: aa rst
Новая строка: aaabbbaaabaaaraaaa rst
Введите подстроку для поиска: ara
Поиск КМП:
Начинается с индекса: 12
Time KMP in nano: 10568800
Стандартный поиск:
Найдено на индексе: 12
Time indexOF in nano: 3700
Process finished with exit code 0
7: Structure
Favorites


"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" "-
Boyer Moore Algorithm Test
Enter Text
arretbus
Enter Pattern
bus
Pattern found at position : 5
Process finished with exit code 0

ЗАДАНИЯ №2

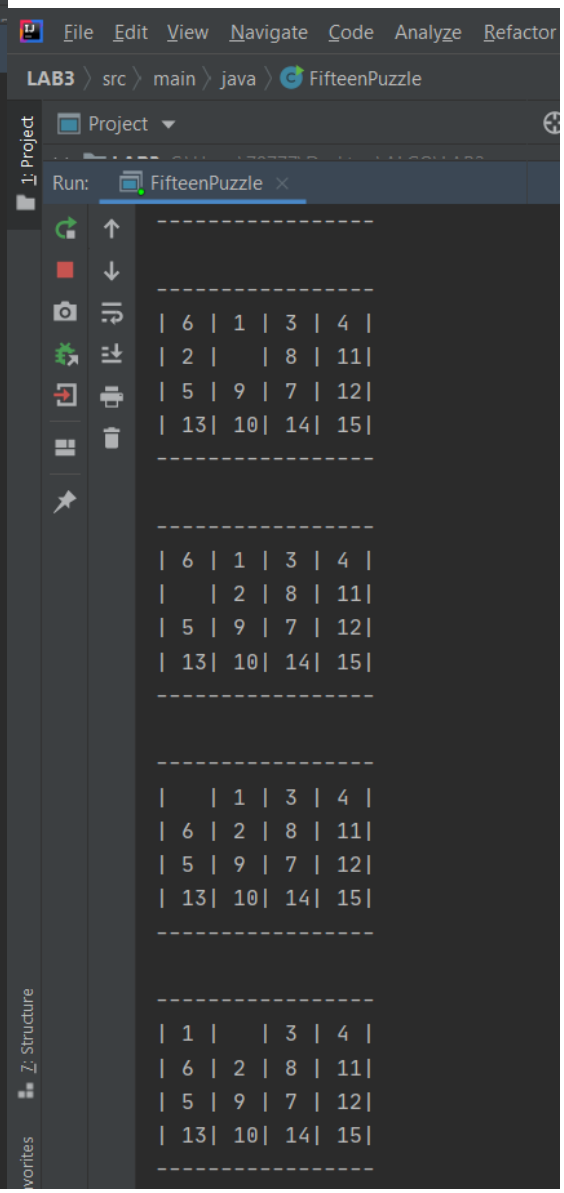
***2**

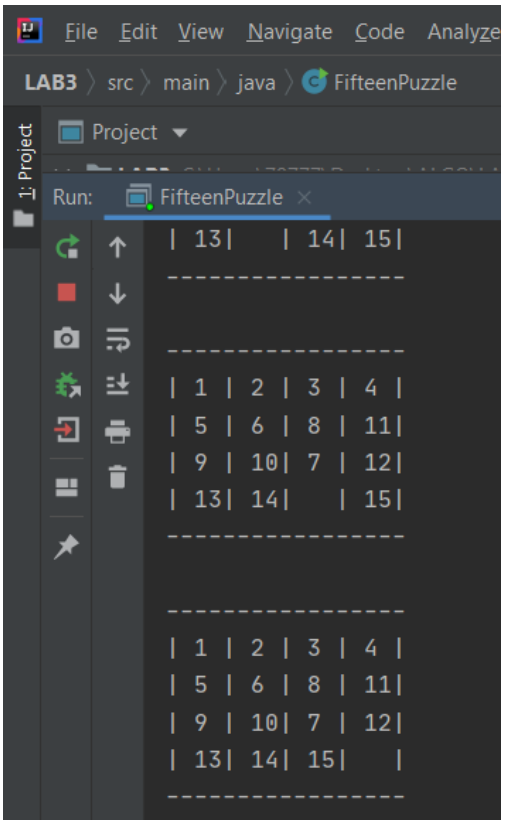

Shuffled board:
3	2	8	4
6	9	1	11
5	7		12
13	10	14	15
Solving with A*
Considered 10,000 positions. Queue = 10,351
Considered 20,000 positions. Queue = 20,330
Considered 30,000 positions. Queue = 30,579
Considered 40,000 positions. Queue = 40,115
Considered 50,000 positions. Queue = 50,044
Considered 60,000 positions. Queue = 60,313
Considered 70,000 positions. Queue = 69,666
Considered 80,000 positions. Queue = 78,900
Considered 90,000 positions. Queue = 89,026
Considered 100,000 positions. Queue = 99,238
Considered 110,000 positions. Queue = 109,251
Considered 120,000 positions. Queue = 118,870
Considered 130,000 positions. Queue = 128,300
Solution considered 130487 boards
Success! Solution with 27 moves:
3	2	8	4
6	9	1	11
5	7		12
13	10	14	15


File Edit View Navigate Code Analyze
LAB3 src main java FifteenPuzzle
Project
Run: FifteenPuzzle
1: Project
7: Structure
Favorites

3	2	8	4
6	9	1	11
5		7	12
13	10	14	15

3	2	8	4
6		1	11
5	9	7	12
13	10	14	15
3		8	4
6	2	1	11
5	9	7	12
13	10	14	15
	3	8	4
6	2	1	11
5	9	7	12
13	10	14	15


File Edit View Navigate Code Analyze Refactor
LAB3 src main java FifteenPuzzle
Project
Run: FifteenPuzzle
1: Project
7: Structure
Favorites
6	1	3	4
2		8	11
5	9	7	12
13	10	14	15
6	1	3	4
	2	8	11
5	9	7	12
13	10	14	15
	1	3	4
6	2	8	11
5	9	7	12
13	10	14	15
1		3	4
6	2	8	11
5	9	7	12
13	10	14	15


File Edit View Navigate Code Analyze
LAB3 src main java FifteenPuzzle
Project
1: Project
Run: FifteenPuzzle
| 13| | 14| 15|

1	2	3	4
5	6	8	11
9	10	7	12
13	14		15

1	2	3	4
5	6	8	11
9	10	7	12
13	14	15	
