```
/*
 * HSR − Uebungen ’Algorithmen & Datenstrukturen 2’
 * Version: Mon Oct 29 17:32:49 CET 2018
 */

package uebung07.as.aufgabe03;

import java.util.Arrays;
import java.util.HashSet;
import java.util.LinkedHashSet;


public class BoyerMoore {

  private static int totCount = 0;
  private static int task = 0;

  public static int bmMatch(String t, int startIndex, int[] last, String p) {

    // TODO Implement here...

    return −1;
  }

  static int[] buildLastFunction(String p) {

    // TODO Implement here...

    return null;
  }

  static void printLastFunction(String t, int[] last) {
    char[] charArr = t.toCharArray();
    Arrays.sort(charArr);
    HashSet<Character> set = new LinkedHashSet<>();
    for (int i = 0; i < charArr.length; i++) {
      set.add(charArr[i]);
    }
    System.out.print("last: ");
    for (char c: set) {
      System.out.print(c+"  ");
    }
    System.out.print("\n    ");
    for (char c: set) {
      System.out.format("%3d", last[c]);
    }
    System.out.println(’\n’);
  }
```



```

  public static void main(String[] args) {
    String text;
    String pattern;
    pattern = text = "";
    if (args.length == 0 || ((args.length == 1) && args[0].equals("1"))) {
      text = "Anna Kurnikowa war eine Tennisspielerin. Sie spielte wieder ein wenig nachdem ihre Beinverletzung fast wieder geheilt war.";
      pattern = "ein";
      task = 1;
    } else if ((args.length == 1) && args[0].equals("2")) {
      text = "adbaacaabedacedbccede";
      pattern = "daeda";
      task = 2;
    } else {
      if (args.length != 2) {
        System.err.println("Bad number of arguments: " + args.length + " (expected: 2)!");
        System.exit(2);
      }
      text = args[0];
      pattern = args[1];
    }
    System.out.println("Text    : "+text);
    System.out.println("Pattern : "+pattern);
    int res = 0;
    int pos = 0;
    int[] last = buildLastFunction(pattern);
    printLastFunction(text, last);

    while (res >= 0) {
      //System.out.println(text.substring(pos));
      res = bmMatch(text, pos, last, pattern);
      if (res >= 0) {
        System.out.println("Position: " + res);
        pos = res + 1;
        if (pos > text.length() − pattern.length()) {
          break;
        }
      }
    }
    System.out.println();
    System.out.println("Total of comparison: " + totCount);
  }
}
```

```
/*
 * HSR − Uebungen ’Algorithmen & Datenstrukturen 2’
 * Version: Mon Oct 29 17:32:49 CET 2018
 */

package uebung07.as.aufgabe03;


public class KnuthMorrisPratt {

  private static int totCount = 0;
  private static int task = 0;

  public static int kmpMatch(String t, int startIndex, int[] fail, String p) {

    // TODO Implement here...

    return −1;
  }

  static int[] buildFailureFunction(String p) {

    // TODO Implement here...

    return null;
  }

  static void printFailureFunction(String p, int[] fail) {
    System.out.print("fail: ");
    for (int i = 0; i < p.length(); i++) {
      System.out.print(p.charAt(i)+"  ");
    }
    System.out.print("\n    ");
    for (int i = 0; i < p.length(); i++) {
      System.out.format("%3d", fail[i]);
    }
    System.out.println(’\n’);
  }
```



```

  public static void main(String[] args) {
    String text;
    String pattern;
    pattern = text = "";
    if (args.length == 0 || ((args.length == 1) && args[0].equals("1"))) {
      text = "Anna Kurnikowa war eine Tennisspielerin. Sie spielte wieder ein wenig nachdem ihre Beinverletzung fast wieder geheilt war.";
      pattern = "ein";
      task = 1;
    } else if ((args.length == 1) && args[0].equals("2")) {
      text = "dcdadaeddaeadaeddadae";
      pattern = "daeda";
      task = 2;
    } else {
      if (args.length != 2) {
        System.err.println("Bad number of arguments: " + args.length + " (expected: 2)!");
        System.exit(2);
      }
      text = args[0];
      pattern = args[1];
    }
    System.out.println("Text    : "+text);
    System.out.println("Pattern : "+pattern);
    int res = 0;
    int pos = 0;
    int[] fail = buildFailureFunction(pattern);
    printFailureFunction(pattern, fail);

    while (res >= 0) {
      //System.out.println(text.substring(pos));
      res = kmpMatch(text, pos, fail, pattern);
      if (res >= 0) {
        pos = res + 1;
        System.out.println("Position: " + res);
      }
    }
    System.out.println();
    System.out.println("Total of comparison: " + totCount);
  }
}
```

```
/*
 * HSR − Uebungen 'Algorithmen & Datenstrukturen 2'
 * Version: Mon Oct 29 17:32:49 CET 2018
 */

package uebung07.as.aufgabe03;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.fail;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import java.util.Random;

import org.junit.Before;
import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class BoyerMooreJUnitTest {

  @Before
  public void setUp() {
    System.setOut(new PrintStream(new ByteArrayOutputStream()));
  }

  @Test
  public void test01BmMatch() {
    String text = "abacaabadcabacabaabb";
    String pattern = "abacab";
    int[] last = BoyerMoore.buildLastFunction(pattern);
    int pos = BoyerMoore.bmMatch(text, 0, last, pattern);
    assertEquals(10, pos);
  }

  @Test
  public void test02StressTestBM() {

    final int NUMBER_OF_TESTS = 10000;

    for (int nr = 0; nr < NUMBER_OF_TESTS; nr++) {
      int textLen = random(1, 500);
      String text = randomText(textLen);
      int patternLen = random(1, 5);
      String pattern = randomText(patternLen);
      int pos = 0;
      while (pos <= textLen − patternLen) {
        int[] last = BoyerMoore.buildLastFunction(pattern);
        int bmPos = BoyerMoore.bmMatch(text, pos, last, pattern);
        int strPos = text.indexOf(pattern, pos);
        if (bmPos != strPos) {
          System.err.println("Bad position : " + bmPos);
          System.err.println("Expected     : " + strPos);
          System.err.println("Text         : >" + text + "<");
          System.err.println("Pattern      : >" + pattern + "<");
          fail("Unexpected result!");
        }
        if (bmPos == −1) {
          break;
        }
        pos = bmPos + 1;
      }
    }
  }
```



```

  /**
   * Returns a random−number in the range from..to.
   *
   * @param from
   *          The lower−bound (inclusive).
   * @param to
   *          The upper−bound (inclusive).
   * @return The generated random−number.
   */
  private int random(int from, int to) {
    return from + (int) (Math.random() * (to − from + 1));
  }

  /**
   * Returns a random−text.
   *
   * @param length
   *          The length of the text to be generated.
   * @return The generated random−text.
   */
  private String randomText(int length) {
    return new Random().ints('a', 'z' + 1)
      .limit(length)
      .collect(StringBuilder::new, (sb, i) −> sb.append((char) i),
       StringBuilder::append)
      .toString();
  }

}
```

```
1  /*
2   * HSR − Uebungen 'Algorithmen & Datenstrukturen 2'
3   * Version: Mon Oct 29 17:32:49 CET 2018
4   */
5
6  package uebung07.as.aufgabe03;
7
8  import static org.junit.Assert.assertEquals;
9  import static org.junit.Assert.fail;
10
11 import java.io.ByteArrayOutputStream;
12 import java.io.PrintStream;
13 import java.util.Random;
14
15 import org.junit.Before;
16 import org.junit.FixMethodOrder;
17 import org.junit.Test;
18 import org.junit.runners.MethodSorters;
19
20 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
21 public class KnuthMorrisPrattJUnitTest {
22
23   @Before
24   public void setUp() {
25     System.setOut(new PrintStream(new ByteArrayOutputStream()));
26   }
27
28   @Test
29   public void test01KmpMatch() {
30     String text = "abacaabaccabacabaabb";
31     String pattern = "abacab";
32     int[] fail = KnuthMorrisPratt.buildFailureFunction(pattern);
33     int pos = KnuthMorrisPratt.kmpMatch(text, 0, fail, pattern);
34     assertEquals(10, pos);
35   }
36
37   @Test
38   public void test02StressTestKMP() {
39
40     final int NUMBER_OF_TESTS = 10000;
41
42     for (int nr = 0; nr < NUMBER_OF_TESTS; nr++) {
43       String text = randomText(1, 500);
44       String pattern = randomText(1, 5);
45       int pos = 0;
46       int kmpPos = 0;
47       while (kmpPos >= 0) {
48         int[] fail = KnuthMorrisPratt.buildFailureFunction(pattern);
49         kmpPos = KnuthMorrisPratt.kmpMatch(text, pos, fail, pattern);
50         int strPos = text.indexOf(pattern, pos);
51         if (kmpPos != strPos) {
52           System.err.println("Bad position : " + kmpPos);
53           System.err.println("Expected     : " + strPos);
54           System.err.println("Text         : >" + text + "<");
55           System.err.println("Pattern      : >" + pattern + "<");
56           fail("Unexpected result!");
57         }
58         pos = kmpPos + 1;
59       }
60     }
61   }
```



```
62
63   /**
64    * Returns a random−text with length in the range min..max.
65    *
66    * @param min
67    *          The lower−bound of length (inclusive).
68    * @param max
69    *          The upper−bound of length (inclusive).
70    * @return The generated random−text.
71    */
72   private String randomText(int min, int max) {
73     int length = min + (int) (Math.random() * (max − min + 1));
74     return new Random().ints('a', 'z' + 1)
75       .limit(length)
76       .collect(StringBuilder::new, (sb, i) −> sb.append((char) i),
77        StringBuilder::append)
78       .toString();
79   }
80
81 }
82
83
84
```