```
/*
 * HSR − Uebungen ’Algorithmen & Datenstrukturen 1’
 * Version: Tue Feb 12 18:21:33 CET 2019
 */

package uebung01.ml.aufgabe02;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public class TextAnalyse {

  public static void main(String[] args) {

    final String text = "Test string to exercise....";
    final char[] chars = { ’a’, ’o’, ’i’, ’e’, ’u’ };

    TextAnalyse textAnalyse = new TextAnalyse();
    int[] charsCounts = textAnalyse.doIt(text.toLowerCase(), chars);
    for (int i = 0; i < chars.length; i++) {
      System.out.println("Output: " + chars[i] + " = " + charsCounts[i]);
    }
    textAnalyse.test();
    textAnalyse.benchmarkTest(chars);
  }

  public int[] doIt(String text, char[] chars) {
    int[] charsCounts = new int[chars.length];

//    // Simple solution:
//    for (int j = 0; j < text.length(); j++) {
//      for (int i = 0; i < chars.length; i++) {
//        if (text.charAt(j) == chars[i]) {
//          charsCounts[i]++;
//          break;
//        }
//      }
//    }

    // Fast solution:
    int[] alfaCounts = new int[’z’+1];
    for (int j = 0; j < text.length(); j++) {
      alfaCounts[text.charAt(j)]++;
    }
    for (int i = 0; i < chars.length; i++) {
      charsCounts[i] = alfaCounts[chars[i]];
    }

    return charsCounts;
  }
```

```

  private void test() {
    System.out.println("\nTesting :\n");
    test(5, 20);

    System.out.print("\nStress−Test : ... ");
    test(100_000, 1_000);
    System.out.println("O.K.");
  }

  private void test(int loops, int textLen) {
    for (int testNr = 0; testNr < loops; testNr++) {
      String text = generateRandomText(textLen, testNr);
      final char[] chars = { ’a’, ’b’, ’c’, ’x’, ’y’, ’z’ };
      int[] charCounts = doIt(text, chars);
      List<Character> textList = text.chars().mapToObj(c −> (char) c)
          .collect(Collectors.toList());
      for (int i = 0; i < chars.length; i++) {
        char c = chars[i];
        int charFrequency = Collections.frequency(textList, c);
        if (charCounts[i] != charFrequency) {
          System.out.println("ERROR");
          System.out.println("Text: " + text);
          System.out.format("Frequency of ’%c’ : %2d\n", c, charFrequency);
          System.out.format("Result of doIt() : %2d\n", charCounts[i]);
          System.exit(1);
        }
      }
      if (textLen <= 20) {
        System.out.println("Text               : " + text);
        System.out.println("Chars to count     : " + Arrays.toString(chars));
        System.out.println("Frequency of chars : " + new String(chars).chars()
            .map(i −> Collections.frequency(textList, (char) i))
            .mapToObj(String::valueOf)
            .collect(Collectors.joining(", ", "[", "]")));
        System.out.println("Result of doIt()   : " + Arrays.toString(charCounts));
      }
    }
  }
```

```java

  @FunctionalInterface
  interface DoIt {
    int[] doIt(String text, char[] chars);
  }

  private void benchmarkTest(char[] chars) {
    System.out.println("\nBenchmark-Test:");
    System.out.print("Recursion : ");
    uebung01.as.aufgabe01.TextAnalyse recursion =
      new uebung01.as.aufgabe01.TextAnalyse();
    long recursionTime = benchmark(recursion::doIt, chars);
    System.out.format("%,7.1f us\n", recursionTime/1000.0);
    System.out.print("Iteration : ");
    TextAnalyse iteration = new TextAnalyse();
    long iterationTime = benchmark(iteration::doIt, chars);
    System.out.format("%,7.1f us\n", iterationTime/1000.0);
    System.out.format("Ratio     : %7.1f\n",
        (Double.valueOf(recursionTime) / iterationTime));
  }

  private long benchmark(DoIt toTest, char[] chars) {
    final int TEXT_LEN = 1_000;
    final int LOOPS = 100_000;
    String text = generateRandomText(TEXT_LEN, 0);
    long startTime, endTime, sumTime = 0;
    try {Thread.sleep(1000);} catch (Exception e) {}
    startTime = System.nanoTime();
    for (int n = 0; n < LOOPS; n++) {
      startTime = System.nanoTime();
      toTest.doIt(text, chars);
      endTime = System.nanoTime();
      sumTime += endTime - startTime;
    }
    return sumTime / LOOPS;
  }

  private String generateRandomText(int len, int seed) {
    return new Random(seed).ints('a', 'z' + 1).limit(len)
        .collect(StringBuilder::new, (sb, i) -> sb.append((char) i),
            StringBuilder::append)
        .toString();
  }

}

```

```

/* Session-Log:

Output: a = 0
Output: o = 1
Output: i = 2
Output: e = 4
Output: u = 0

Testing :

Text               : ssxvnjhpdqdxvcrastvy
Chars to count     : [a, b, c, x, y, z]
Frequency of chars : [1, 0, 1, 2, 1, 0]
Result of doIt()   : [1, 0, 1, 2, 1, 0]
Text               : rahjmyuwwkrxnfmqgeeb
Chars to count     : [a, b, c, x, y, z]
Frequency of chars : [1, 1, 0, 1, 1, 0]
Result of doIt()   : [1, 1, 0, 1, 1, 0]
Text               : sgavreiznecykaewpvcr
Chars to count     : [a, b, c, x, y, z]
Frequency of chars : [2, 0, 2, 0, 1, 1]
Result of doIt()   : [2, 0, 2, 0, 1, 1]
Text               : smmhquvgjxpybczmehmt
Chars to count     : [a, b, c, x, y, z]
Frequency of chars : [0, 1, 1, 1, 1, 1]
Result of doIt()   : [0, 1, 1, 1, 1, 1]
Text               : qsnwfpfwipeusiwkzogm
Chars to count     : [a, b, c, x, y, z]
Frequency of chars : [0, 0, 0, 0, 0, 1]
Result of doIt()   : [0, 0, 0, 0, 0, 1]

Stress-Test : ... O.K.

Benchmark-Test:
Recursion :    19.1 us
Iteration :     1.1 us
Ratio     :    17.4

*/

```