Nama : Andyan Yogawardhana

NIM : 21/482180/PA/21030

Kelas : KOMB1

Tugas 10 – String Matching

1. Diberikan sebuah teks = "aaaa...aa" (100.000 buah huruf a) dan sebuah pola = "aaa...aab" (10.000 buah huruf a dan 1 huruf b). Bandingkan running time dari Algoritma naïve dan Algoritma KMP untuk pencocokan string! Lakukan pencocokan string 10 kali, dan selanjutnya hitung rata-rata running time untuk masing-masing algoritma.

Source Code

```
1  public class StringMatching1 {
2      public static void main(String[] args) {
3          Scanner sc = new Scanner(System.in);
4          String text, pattern;
5          // array of char untuk menampung karakter yang akan di-
   assign secara repetitif dengan memanfaatkan loop
6          char[] textChar = new char[100000];     // teks
   sepanjang 100.000 buah huruf a
7          char[] patternChar = new char[10001];   // pola
   sepanjang 10.000 buah huruf a dan 1 huruf b
8
9          // System.out.print("Input text: ");
10         // text = sc.nextLine();
11
12         // melakukan assign karakter sesuai perintah soal dengan
   loop
13         for(int i = 0; i < 100000; i++) {
14             textChar[i] = 'a';      // "100.000 buah huruf a"
15         }
16         // melakukan casting tipe data dari array of char
   menjadi string
17         text = new String(textChar);
18
19         // melakukan assign karakter sesuai perintah soal dengan
   loop
20         for(int i = 0; i < 10000; i++) {
21             patternChar[i] = 'a';   // "10.000 buah huruf a"
22         }
23         patternChar[10000] = 'b';   // "satu huruf b"
24         // melakukan casting tipe data dari array of char
   menjadi string
```

```java
        pattern = new String(patternChar);

        // System.out.print("Input pattern: ");
        // do pattern = sc.nextLine();
        // while (pattern.equals(""));

        StringMatcher.naive(text, pattern);
        StringMatcher.kmp(text, pattern);

        sc.close();
    }
}

class StringMatcher {
    public static void naive(String text, String pattern) {
        // memulai penghitungan running time dengan fungsi bawaan currentTimeMillis yang mengembalikan nilai waktu dalam milisekon
        long startTime = System.currentTimeMillis();
        // melakukan loop untuk mencocokkan string sebanyak 10 kali
        for(int repeat = 0; repeat < 10; repeat++) {
            int textLen = text.length();
            int patternLen = pattern.length();

            boolean found = false;

            for(int i = 0; i + patternLen <= textLen; i++) {
                boolean currentFound = true;
                for(int j = 0; j < patternLen; j++) {
                    if(text.charAt(i + j) != pattern.charAt(j)) {
                        currentFound = false;
                        break;
                    }
                }
                if(currentFound) {
                    found = true;
                    System.out.println("Found pattern at index " + i + " using naive.");
                }
            }
            if(!found) {
                System.out.println("Pattern not found using naive.");
            }
        }
        // akhir penghitungan running time
```

```
67          long stopTime = System.currentTimeMillis();
68          // melakukan pencetakan hasil penghitungan running time
69          System.out.println("Average Naive running time: " +
    ((stopTime - startTime)/10) + " milisecond(s)");
70      }
71
72      private static int[] computeLPSArray(String str) {
73          int len = str.length();
74
75          int[] lps = new int[len];
76          lps[0] = 0;
77
78          for(int i = 1; i < len; i++) {
79              int j = lps[i - 1];
80
81              while((j > 0) && (str.charAt(i) != str.charAt(j))) {
82                  j = lps[j - 1];
83              }
84
85              if(str.charAt(i) == str.charAt(j)) {
86                  j++;
87              }
88
89              lps[i] = j;
90          }
91
92          return lps;
93      }
94
95      public static void kmp(String text, String pattern) {
96          // memulai penghitungan running time dengan fungsi
    bawaan currentTimeMillis yang mengembalikan nilai waktu dalam
    milisekon
97          long startTime = System.currentTimeMillis();
98          // melakukan loop untuk mencocokkan string sebanyak 10
    kali
99          for(int repeat = 0; repeat < 10; repeat++) {
100                 String combined = pattern + '#' + text;
101                 int combinedLen = combined.length();
102                 int patternLen = pattern.length();
103
104                 int lps[] = computeLPSArray(combined);
105
106                 boolean found = false;
107
108                 for(int i = patternLen + 1; i < combinedLen;
    i++) {
109                     if(lps[i] == patternLen) {
```

```
110                         found = true;
111                         System.out.println("Found pattern at
   index " + (i - 2 * patternLen) + " using KMP.");
112                     }
113                 }
114 
115                 if(!found) {
116                     System.out.println("Pattern not found
   using KMP.");
117                 }
118             }
119             // akhir penghitungan running time
120             long stopTime = System.currentTimeMillis();
121             // melakukan pencetakan hasil penghitungan running
   time
122             System.out.println("Average KMP running time  : "
   + ((stopTime - startTime)/10) + " milisecond(s)");
123         }
124     }
```

Output Terminal

```
Pattern not found using naive.
Pattern not found using naive.
Pattern not found using naive.
Pattern not found using naive.
Pattern not found using naive.
Pattern not found using naive.
Pattern not found using naive.
Pattern not found using naive.
Pattern not found using naive.
Pattern not found using naive.
Average Naive running time: 6532 milisecond(s)
Pattern not found using KMP.
Pattern not found using KMP.
Pattern not found using KMP.
Pattern not found using KMP.
Pattern not found using KMP.
Pattern not found using KMP.
Pattern not found using KMP.
Pattern not found using KMP.
Pattern not found using KMP.
Pattern not found using KMP.
Average KMP running time  : 9 milisecond(s)
```

2. Diberikan sebuah teks = "aaaa...aa" (100.000 buah huruf a) dan sebuah pola = "aaa...aa" (10.000 buah huruf a). Bandingkan running time dari Algoritma naïve dan Algoritma KMP (tanpa output apapun) untuk pencocokan string! Lakukan pencocokan string 10 kali, dan selanjutnya hitung rata-rata running time untuk masing-masing algoritma.

Source Code

```java
public class StringMatching2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String text, pattern;
        // array of char untuk menampung karakter yang akan di-assign secara repetitif dengan memanfaatkan loop
        char[] textChar = new char[100000];     // teks sepanjang 100.000 buah huruf a
        char[] patternChar = new char[10000];   // pola sepanjang 10.000 buah huruf a

        // System.out.print("Input text: ");
        // text = sc.nextLine();

        // melakukan assign karakter sesuai perintah soal dengan loop
        for(int i = 0; i < 100000; i++) {
            textChar[i] = 'a';      // "100.000 buah huruf a"
        }
        // melakukan casting tipe data dari array of char menjadi string
        text = new String(textChar);

        // melakukan assign karakter sesuai perintah soal dengan loop
        for(int i = 0; i < 10000; i++) {
            patternChar[i] = 'a';   // "10.000 buah huruf a"
        }
        // melakukan casting tipe data dari array of char menjadi string
        pattern = new String(patternChar);

        // System.out.print("Input pattern: ");
        // do pattern = sc.nextLine();
        // while (pattern.equals(""));

        StringMatcher.naive(text, pattern);
        StringMatcher.kmp(text, pattern);

        sc.close();
```

```
    }
}

class StringMatcher {
    public static void naive(String text, String pattern) {
        // memulai penghitungan running time dengan fungsi bawaan currentTimeMillis yang mengembalikan nilai waktu dalam milisekon
        long startTime = System.currentTimeMillis();
        // melakukan loop untuk mencocokkan string sebanyak 10 kali
        for(int repeat = 0; repeat < 10; repeat++) {
            int textLen = text.length();
            int patternLen = pattern.length();

            boolean found = false;

            for(int i = 0; i + patternLen <= textLen; i++) {
                boolean currentFound = true;
                for(int j = 0; j < patternLen; j++) {
                    if(text.charAt(i + j) != pattern.charAt(j)) {
                        currentFound = false;
                        break;
                    }
                }
                if(currentFound) {
                    found = true;
                    // System.out.println("Found pattern at index " + i + " using naive.");
                }
            }
            // if(!found) {
                // System.out.println("Pattern not found using naive.");
            // }
        }
        // akhir penghitungan running time
        long stopTime = System.currentTimeMillis();
        // melakukan pencetakan hasil penghitungan running time
        System.out.println("Average Naive running time: " + ((stopTime - startTime)/10) + " milisecond(s)");
    }

    private static int[] computeLPSArray(String str) {
        int len = str.length();

        int[] lps = new int[len];
```

```
75          lps[0] = 0;
76
77          for(int i = 1; i < len; i++) {
78              int j = lps[i - 1];
79
80              while((j > 0) && (str.charAt(i) != str.charAt(j))) {
81                  j = lps[j - 1];
82              }
83
84              if(str.charAt(i) == str.charAt(j)) {
85                  j++;
86              }
87
88              lps[i] = j;
89          }
90
91          return lps;
92      }
93
94      public static void kmp(String text, String pattern) {
95          // memulai penghitungan running time dengan fungsi
    bawaan currentTimeMillis yang mengembalikan nilai waktu dalam
    milisekon
96          long startTime = System.currentTimeMillis();
97          // melakukan loop untuk mencocokkan string sebanyak 10
    kali
98          for(int repeat = 0; repeat < 10; repeat++) {
99              String combined = pattern + '#' + text;
100                 int combinedLen = combined.length();
101                 int patternLen = pattern.length();
102
103                 int lps[] = computeLPSArray(combined);
104
105                 boolean found = false;
106
107                 for(int i = patternLen + 1; i < combinedLen;
    i++) {
108                     if(lps[i] == patternLen) {
109                         found = true;
110                         // System.out.println("Found pattern
    at index " + (i - 2 * patternLen) + " using KMP.");
111                     }
112                 }
113
114                 // if(!found) {
115                     // System.out.println("Pattern not found
    using KMP.");
116                 // }
```

```
117             }
118             // akhir penghitungan running time
119             long stopTime = System.currentTimeMillis();
120             // melakukan pencetakan hasil penghitungan running
    time
121             System.out.println("Average KMP running time  : "
    + ((stopTime - startTime)/10) + " milisecond(s)");
122         }
123     }
```

Output Terminal

```
Average Naive running time: 4578 milisecond(s)
Average KMP running time  : 7 milisecond(s)
```

3. Diberikan dua kata S dan T, dengan panjang yang sama (panjang maksimal adalah 100.000). Tugas Anda adalah menentukan apakah T dapat dibuat dengan melakukan beberapa cycle shift ke S (cycle shift adalah pemindahan karakter pertama string ke akhir string). Misalnya jika S = "erwineko" dan T = "ekoerwin", maka jawabannya haruslah "YA", karena "erwineko" -> "rwinekoe" -> "winekoer" -> "inekoerw" -> "nekoerwi" -> "ekoerwin". Dalam soal ini, Anda harus menggunakan Algoritma KMP untuk menyelesaikan soal ini (pendekatannya mungkin tidak begitu jelas, tetapi Anda harus memikirkan penggunaan Algoritma KMP dalam masalah ini).

Source Code

```java
import java.util.Scanner;

public class StringMatching3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String text, pattern;

        System.out.print("Input text    : ");
        text = sc.nextLine();

        System.out.print("Input pattern : ");
        do pattern = sc.nextLine();
        while (pattern.equals(""));

        StringMatcher.kmp(text, pattern);

        sc.close();
    }
}

class StringMatcher {
    private static int[] computeLPSArray(String str) {
        int len = str.length();

        int[] lps = new int[len];
        lps[0] = 0;

        for(int i = 1; i < len; i++) {
            int j = lps[i - 1];

            while((j > 0) && (str.charAt(i) != str.charAt(j))) {
                j = lps[j - 1];
            }

```

```java
            if(str.charAt(i) == str.charAt(j)) {
                j++;
            }

            lps[i] = j;
        }

        return lps;
    }

    public static void kmp(String text, String pattern) {
        String combined = pattern + '#' + text + text;  // + text untuk menemukan pola yang berulang dalam suatu text
        // digunakan dalam print cycle shift
        String combinedText = text + text;
        int combinedLen = combined.length();
        int patternLen = pattern.length();
        // menampung index awal text yang memiliki pattern
        int indexFound = 0;
        int lps[] = computeLPSArray(combined);

        boolean found = false;

        for(int i = patternLen + 1; i < combinedLen; i++) {
            if(lps[i] == patternLen) {
                found = true;
                // assignment nilai index pattern di text
                indexFound = i - 2 * patternLen;
                System.out.println("Found pattern at index " + (i - 2 * patternLen) + " using KMP.");
            }
        }

        if(found) {
            // print proses cycle shift
            System.out.print("Cycle shift   : " + text);
            // melakukan loop cycle shift dari text input hingga menjadi pattern input
            for(int i = 1; i <= indexFound; i++) {
                System.out.print(" -> " + combinedText.substring(i, i + patternLen));
            }
        }
        else {
            System.out.println("Pattern not found using KMP.");
        }
    }
}
```

Output Terminal

```
Input text    : erwineko
Input pattern : ekoerwin
Found pattern at index 5 using KMP.
Cycle shift   : erwineko -> rwinekoe -> winekoer -> inekoerw -> nekoerwi -> ekoerwin
```