Nama : Andyan Yogawardhana

NIM : 21/482180/PA/21030

Kelas : KOMB1

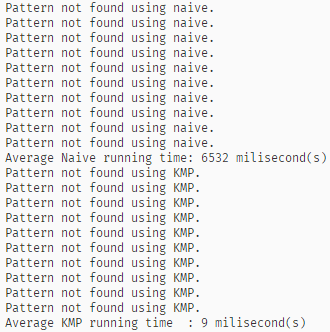
Tugas 10 – String Matching

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. *// System.out.print("Input text: ");*
9. *// text = sc.nextLine();*
10. *// melakukan assign karakter sesuai perintah soal dengan loop*
11. for(int i = 0; i < 100000; i++) {
12. textChar[i] = 'a';      *// "100.000 buah huruf a"*
13. }
14. *// melakukan casting tipe data dari array of char menjadi string*
15. text = new String(textChar);
17. *// melakukan assign karakter sesuai perintah soal dengan loop*
18. for(int i = 0; i < 10000; i++) {
19. patternChar[i] = 'a';   *// "10.000 buah huruf a"*
20. }
21. patternChar[10000] = 'b';   *// "satu huruf b"*
22. *// melakukan casting tipe data dari array of char menjadi string*
23. pattern = new String(patternChar);
24. *// System.out.print("Input pattern: ");*
25. *// do pattern = sc.nextLine();*
26. *// while (pattern.equals(""));*
27. StringMatcher.naive(text, pattern);
28. StringMatcher.kmp(text, pattern);
29. sc.close();
30. }
31. }
32. class StringMatcher {
33. public static void naive(String text, String pattern) {
34. *// memulai penghitungan running time dengan fungsi bawaan currentTimeMillis yang mengembalikan nilai waktu dalam milisekon*
35. long startTime = System.currentTimeMillis();
36. *// melakukan loop untuk mencocokkan string sebanyak 10 kali*
37. for(int repeat = 0; repeat < 10; repeat++) {
38. int textLen = text.length();
39. int patternLen = pattern.length();
41. boolean found = false;
43. for(int i = 0; i + patternLen <= textLen; i++) {
44. boolean currentFound = true;
45. for(int j = 0; j < patternLen; j++) {
46. if(text.charAt(i + j) != pattern.charAt(j)) {
47. currentFound = false;
48. break;
49. }
50. }
51. if(currentFound) {
52. found = true;
53. System.out.println("Found pattern at index " + i + " using naive.");
54. }
55. }
56. if(!found) {
57. System.out.println("Pattern not found using naive.");
58. }
59. }
60. *// akhir penghitungan running time*
61. long stopTime = System.currentTimeMillis();
62. *// melakukan pencetakan hasil penghitungan running time*
63. System.out.println("Average Naive running time: " + ((stopTime - startTime)/10) + " milisecond(s)");
64. }
65. private static int[] computeLPSArray(String str) {
66. int len = str.length();
67. int[] lps = new int[len];
68. lps[0] = 0;
69. for(int i = 1; i < len; i++) {
70. int j = lps[i - 1];
71. while((j > 0) && (str.charAt(i) != str.charAt(j))) {
72. j = lps[j - 1];
73. }
74. if(str.charAt(i) == str.charAt(j)) {
75. j++;
76. }
77. lps[i] = j;
78. }
79. return lps;
80. }
81. public static void kmp(String text, String pattern) {
82. *// memulai penghitungan running time dengan fungsi bawaan currentTimeMillis yang mengembalikan nilai waktu dalam milisekon*
83. long startTime = System.currentTimeMillis();
84. *// melakukan loop untuk mencocokkan string sebanyak 10 kali*
85. for(int repeat = 0; repeat < 10; repeat++) {
86. String combined = pattern + '#' + text;
87. int combinedLen = combined.length();
88. int patternLen = pattern.length();
89. int lps[] = computeLPSArray(combined);
90. boolean found = false;
92. for(int i = patternLen + 1; i < combinedLen; i++) {
93. if(lps[i] == patternLen) {
94. found = true;
95. System.out.println("Found pattern at index " + (i - 2 \* patternLen) + " using KMP.");
96. }
97. }
98. if(!found) {
99. System.out.println("Pattern not found using KMP.");
100. }
101. }
102. *// akhir penghitungan running time*
103. long stopTime = System.currentTimeMillis();
104. *// melakukan pencetakan hasil penghitungan running time*
105. System.out.println("Average KMP running time  : " + ((stopTime - startTime)/10) + " milisecond(s)");
106. }
107. }

Output Terminal



1. 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

1. public class StringMatching2 {
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[10000];   *// pola sepanjang 10.000 buah huruf a*
8. *// System.out.print("Input text: ");*
9. *// text = sc.nextLine();*
10. *// melakukan assign karakter sesuai perintah soal dengan loop*
11. for(int i = 0; i < 100000; i++) {
12. textChar[i] = 'a';      *// "100.000 buah huruf a"*
13. }
14. *// melakukan casting tipe data dari array of char menjadi string*
15. text = new String(textChar);
17. *// melakukan assign karakter sesuai perintah soal dengan loop*
18. for(int i = 0; i < 10000; i++) {
19. patternChar[i] = 'a';   *// "10.000 buah huruf a"*
20. }
21. *// melakukan casting tipe data dari array of char menjadi string*
22. pattern = new String(patternChar);
23. *// System.out.print("Input pattern: ");*
24. *// do pattern = sc.nextLine();*
25. *// while (pattern.equals(""));*
26. StringMatcher.naive(text, pattern);
27. StringMatcher.kmp(text, pattern);
28. sc.close();
29. }
30. }
31. class StringMatcher {
32. public static void naive(String text, String pattern) {
33. *// memulai penghitungan running time dengan fungsi bawaan currentTimeMillis yang mengembalikan nilai waktu dalam milisekon*
34. long startTime = System.currentTimeMillis();
35. *// melakukan loop untuk mencocokkan string sebanyak 10 kali*
36. for(int repeat = 0; repeat < 10; repeat++) {
37. int textLen = text.length();
38. int patternLen = pattern.length();
40. boolean found = false;
42. for(int i = 0; i + patternLen <= textLen; i++) {
43. boolean currentFound = true;
44. for(int j = 0; j < patternLen; j++) {
45. if(text.charAt(i + j) != pattern.charAt(j)) {
46. currentFound = false;
47. break;
48. }
49. }
50. if(currentFound) {
51. found = true;
52. *// System.out.println("Found pattern at index " + i + " using naive.");*
53. }
54. }
55. *// if(!found) {*
56. *// System.out.println("Pattern not found using naive.");*
57. *// }*
58. }
59. *// akhir penghitungan running time*
60. long stopTime = System.currentTimeMillis();
61. *// melakukan pencetakan hasil penghitungan running time*
62. System.out.println("Average Naive running time: " + ((stopTime - startTime)/10) + " milisecond(s)");
63. }
64. private static int[] computeLPSArray(String str) {
65. int len = str.length();
66. int[] lps = new int[len];
67. lps[0] = 0;
68. for(int i = 1; i < len; i++) {
69. int j = lps[i - 1];
70. while((j > 0) && (str.charAt(i) != str.charAt(j))) {
71. j = lps[j - 1];
72. }
73. if(str.charAt(i) == str.charAt(j)) {
74. j++;
75. }
76. lps[i] = j;
77. }
78. return lps;
79. }
80. public static void kmp(String text, String pattern) {
81. *// memulai penghitungan running time dengan fungsi bawaan currentTimeMillis yang mengembalikan nilai waktu dalam milisekon*
82. long startTime = System.currentTimeMillis();
83. *// melakukan loop untuk mencocokkan string sebanyak 10 kali*
84. for(int repeat = 0; repeat < 10; repeat++) {
85. String combined = pattern + '#' + text;
86. int combinedLen = combined.length();
87. int patternLen = pattern.length();
88. int lps[] = computeLPSArray(combined);
89. boolean found = false;
91. for(int i = patternLen + 1; i < combinedLen; i++) {
92. if(lps[i] == patternLen) {
93. found = true;
94. *// System.out.println("Found pattern at index " + (i - 2 \* patternLen) + " using KMP.");*
95. }
96. }
97. *// if(!found) {*
98. *// System.out.println("Pattern not found using KMP.");*
99. *// }*
100. }
101. *// akhir penghitungan running time*
102. long stopTime = System.currentTimeMillis();
103. *// melakukan pencetakan hasil penghitungan running time*
104. System.out.println("Average KMP running time  : " + ((stopTime - startTime)/10) + " milisecond(s)");
105. }
106. }

Output Terminal



1. 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

1. import java.util.Scanner;
2. public class StringMatching3 {
3. public static void main(String[] args) {
4. Scanner sc = new Scanner(System.in);
5. String text, pattern;
6. System.out.print("Input text    : ");
7. text = sc.nextLine();
8. System.out.print("Input pattern : ");
9. do pattern = sc.nextLine();
10. while (pattern.equals(""));
11. StringMatcher.kmp(text, pattern);
12. sc.close();
13. }
14. }
15. class StringMatcher {
16. private static int[] computeLPSArray(String str) {
17. int len = str.length();
18. int[] lps = new int[len];
19. lps[0] = 0;
20. for(int i = 1; i < len; i++) {
21. int j = lps[i - 1];
22. while((j > 0) && (str.charAt(i) != str.charAt(j))) {
23. j = lps[j - 1];
24. }
25. if(str.charAt(i) == str.charAt(j)) {
26. j++;
27. }
28. lps[i] = j;
29. }
30. return lps;
31. }
32. public static void kmp(String text, String pattern) {
33. String combined = pattern + '#' + text + text;  *// + text untuk menemukan pola yang berulang dalam suatu text*
34. *// digunakan dalam print cycle shift*
35. String combinedText = text + text;
36. int combinedLen = combined.length();
37. int patternLen = pattern.length();
38. *// menampung index awal text yang memiliki pattern*
39. int indexFound = 0;
40. int lps[] = computeLPSArray(combined);
41. boolean found = false;
43. for(int i = patternLen + 1; i < combinedLen; i++) {
44. if(lps[i] == patternLen) {
45. found = true;
46. *// assignment nilai index pattern di text*
47. indexFound = i - 2 \* patternLen;
48. System.out.println("Found pattern at index " + (i - 2 \* patternLen) + " using KMP.");
49. }
50. }
51. if(found) {
52. *// print proses cycle shift*
53. System.out.print("Cycle shift   : " + text);
54. *// melakukan loop cycle shift dari text input hingga menjadi pattern input*
55. for(int i = 1; i <= indexFound; i++) {
56. System.out.print(" -> " + combinedText.substring(i, i + patternLen));
57. }
58. }
59. else {
60. System.out.println("Pattern not found using KMP.");
61. }
62. }
63. }

Output Terminal

