Day-11 Assignment

ManjulaNanuri

Task 1: String Operations

Write a method that takes two strings, concatenates them, reverses the result, and then extracts the middle substring of the given length. Ensure your method handles edge cases, such as an empty string or a substring length larger than the concatenated string.

```
package com.example.string;

public class StringOperations {

public static String manipulateStrings(String str1, String str2, int length) {

String concatenated = str1 + str2;

String reversed = new StringBuilder(concatenated).reverse().toString();

if (length >= reversed.length()) {

return reversed;

}

if (length <= 0) {

return "";

}

int startIndex = (reversed.length() - length) / 2;

return reversed.substring(startIndex, startIndex + length);

}

public static void main(String[] args) {

System.out.println(manipulateStrings("hello", "world", 5));

System.out.println(manipulateStrings("hello", "", 3));

System.out.println(manipulateStrings("", "world", 4));
System.out.println(manipulateStrings("a", "b", 10));

System.out.println(manipulateStrings("abc", "def", 0));
```

```
 }

}
```

OUTPUT:

```
<terminated> StringOperations [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (06-Jun-202
rowol
lle
dlro
ba
```

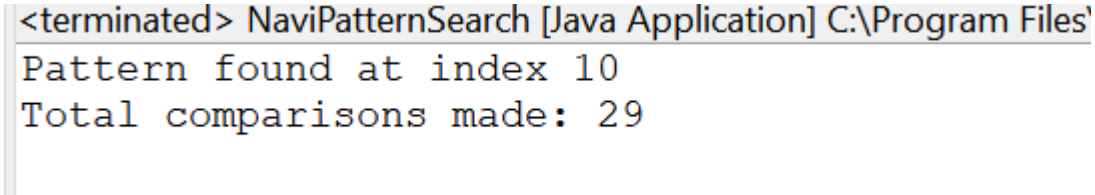
Task 2: Naive Pattern Search

Implement the naive pattern searching algorithm to find all occurrences of a pattern within a given text string. Count the number of comparisons made during the search to evaluate the efficiency of the algorithm.

```
package com.example.string;

public class NaviPatternSearch {

public static void naivePatternSearch(String text, String pattern) {

int textLength = text.length();

int patternLength = pattern.length();

int comparisons = 0;

for (int i = 0; i <= textLength - patternLength; i++) {

int j;

for (j = 0; j < patternLength; j++) {

comparisons++;

if (text.charAt(i + j) != pattern.charAt(j)) {

break;

}

}

if (j == patternLength) {
```

```
System.out.println("Pattern found at index " + i);

}

}

System.out.println("Total comparisons made: " + comparisons);

}

public static void main(String[] args) {

String text = "ABABDABACDABABCABAB";

String pattern = "ABABCABAB";

naivePatternSearch(text, pattern);

}

}
```

OUTPUT:



Task 3: Implementing the KMP Algorithm

Code the Knuth-Morris-Pratt (KMP) algorithm in C# for pattern searching which pre-processes the pattern to reduce the number of comparisons. Explain how this pre-processing improves the search time compared to the naive approach.

```
package com.example.string;

public class KMPAlgorithm {

private static int[] ComputeLPSArray(String pattern) {

int length = 0;

int i = 1;
```

```
int[] lps = new int[pattern.length()];

lps[0] = 0;

while (i < pattern.length()) {

if (pattern.charAt(i) == pattern.charAt(length)) {

length++;

lps[i] = length;

i++;

} else {

if (length != 0) {

length = lps[length - 1];

} else {

lps[i] = 0;

i++;

}

}

}

return lps;

}

// Function to perform KMP search

public static void KMPSearch(String text, String pattern) {

int[] lps = ComputeLPSArray(pattern);

int i = 0;

int j = 0;

while (i < text.length()) {
```

```
if (pattern.charAt(j) == text.charAt(i)) {
i++;
j++;
}
if (j == pattern.length()) {
System.out.println("Pattern found at index " + (i - j));
j = lps[j - 1];
} else if (i < text.length() && pattern.charAt(j) != text.charAt(i)) {
if (j != 0) {
j = lps[j - 1];
} else {
i++;
}
}
}
}
public static void main(String[] args) {
String text = "ABABDABACDABABCABAB";
String pattern = "ABABCABAB";
KMPSearch(text, pattern);
}
}
```

OUTPUT:

```
<terminated> KMPAlgorithm [Java Application] C:\Program Files\Java
Pattern found at index 10
```

Task 4: Rabin-Karp Substring Search

Implement the Rabin-Karp algorithm for substring search using a rolling hash. Discuss the impact of hash collisions on the algorithm's performance and how to handle them.

```
package com.example.string;

public class RabinKarpSubstring {

public static final int d = 256;

public static final int q = 101;

public static void RabinKarpSearch(String text, String pattern) {

int m = pattern.length();

int n = text.length();

int i, j;

int p = 0;

int t = 0;

int h = 1;

for (i = 0; i < m - 1; i++)

h = (h * d) % q;

for (i = 0; i < m; i++) {

p = (d * p + pattern.charAt(i)) % q;

t = (d * t + text.charAt(i)) % q;

}

for (i = 0; i <= n - m; i++) {

if (p == t) {
```

```
for (j = 0; j < m; j++) {

if (text.charAt(i + j) != pattern.charAt(j))

break;

}

if (j == m)

System.out.println("Pattern found at index " + i);

}if (i < n - m) {

t = (d * (t - text.charAt(i) * h) + text.charAt(i + m)) % q;

if (t < 0)

t = (t + q);

}

}}

public static void main(String[] args) {

String text = "ABCCDDAEFG";

String pattern = "CDD";

RabinKarpSearch(text, pattern);

}}
```

OUTPUT:

```
<terminated> RabinKarpSubstring [Java Application] C:\Program Files
Pattern found at index 3
```

Task 5: Boyer-Moore Algorithm Application

Use the Boyer-Moore algorithm to write a function that finds the last occurrence of a substring in a given string and returns its index. Explain why this algorithm can outperform others in certain scenarios.

```
package com.example.string;

import java.util.Arrays;

public class BoyerMooreSubstring {

private static int[] preprocessBadCharacterHeuristic(String pattern) {

int[] badChar = new int[256];

Arrays.fill(badChar, -1);

for (int i = 0; i < pattern.length(); i++) {

badChar[pattern.charAt(i)] = i;

}

return badChar;

}

public static int findLastOccurrence(String text, String pattern) {

int[] badChar = preprocessBadCharacterHeuristic(pattern);

int m = pattern.length();

int n = text.length();

int shift = 0;

int lastOccurrenceIndex = -1;

while (shift <= (n - m)) {

int j = m - 1;

while (j >= 0 && pattern.charAt(j) == text.charAt(shift + j)) {

j--;

}
```

```
if (j < 0) {
lastOccurrenceIndex = shift;
shift += (shift + m < n) ? m - badChar[text.charAt(shift + m)] : 1;
} else {
shift += Math.max(1, j - badChar[text.charAt(shift + j)]);
}
}
return lastOccurrenceIndex;
}
public static void main(String[] args) {
String text = "ABAAABCDABCABC";
String pattern = "ABC";
int index = findLastOccurrence(text, pattern);
if (index != -1) {
System.out.println("Last occurrence of pattern is at index: " + index);
} else {
System.out.println("Pattern not found in the text.");
}
}
}
```

OUTPUT:

```
<terminated> BoyerMooreSubstring [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe
Last occurrence of pattern is at index: 11
```