LAB_11

ICS 202

Jalal Ali Zainaddin | 202154790 | 12/06/2023

Task 1: find Longest non-overlapping suffix that is also a prefix:

Declare Prefix and LPS, then run one loop to iterate through all characters, each time we compare their prefix and suffix, if they are equal, assign LPS string to the longest common.

Code:

```java
// task 1
1 usage
public static String findLongestPrefixSuffix(String str) {
    String prefix = "";
    String LPS = "";

    for (int i = 0; i < str.length()/2; i++) {
        // Add the current character to the prefix string
        prefix += str.charAt(i);
        // Store the suffix string
        String suffix = str.substring(str.length() - 1 - i, str.length());

        // Check if both the strings are equal or not
        if (prefix.equals(suffix))
            LPS = prefix;
    }
    return LPS;
}
```

Output:

```
Enter a pattern to search for: ABABCABAB
Longest non-overlapping suffix that is also a prefix is: ABAB its length is: 4
Enter a pattern to search for: ABCDE
No non-overlapping suffix that is also a prefix.
```

Task 2: brute force string matching algorithm:

First we check if the pattern is already exist anywhere, if yes, then we iterate through all characters, each time we start we the next, if the next characters equals to the pattern, print, else skip to next.

Code:

```java
// task 2
1 usage
public static void overlappingPatternSearch(String T, String P){
    //check if pattern exist anywhere
    if (T.indexOf(P) == -1){
        System.out.println("Pattern not found.");
        return;
    }

    for (int i = 0; i < T.length()-P.length()+1; i++) {
        // check if pattern equals to the next strings that has the same length
        if (P.equals(T.substring(i,P.length()+i))){
            //print
            System.out.println(T);
            System.out.println(" ".repeat(i)+P);
            System.out.println(" ".repeat(i)+i);
        }
    }

}
```

Output:

```
Enter a text string T: ABABABCDABABK
Enter a pattern string P: ABAB
ABABABCDABABK
ABAB
0
ABABABCDABABK
  ABAB
  2
ABABABCDABABK
        ABAB
        8
```

```
Enter a text string T: THIS IS KFUPM
Enter a pattern string P: YES
Pattern not found.
```

Task 3.1: generate all proper overlapping suffixes and prefixes of the string:

First we declare a prefix and a current string for the suffix, we have two loop, each time the first one run, it stores the current string from the first to the current index, then we run a second loop to add every individual character to prefix and suffix, then compare them if they are equal or not, if they are print the prefix and its length.

Code:

```java
// task 3
1 usage
static void countSamePrefixSuffix(String s, int n)
{
    // Stores the prefix string
    String prefix = "";
    for (int i = 0; i < n+1; i++) {
        // store a current string
        String current = s.substring(0,i);
        System.out.println("Substring: " + s.substring(0,i));
        prefix = "";
        for (int j = 0; j < i-1; j++) {
            // Add the current character to the prefix string
            prefix += s.charAt(j);

            // Store the suffix string
            String suffix = current.substring(current.length()-j-1, current.length());

            System.out.print("Proper prefix: " + prefix + ", Proper suffix: " + suffix);

            // Check if both the strings are equal or not
            if (prefix.equals(suffix)) {
                System.out.print(" *" + prefix.length());
            }

            System.out.println();
        }
    }
}
```

Output:

```
Substring:
------------------------------
Substring: A
------------------------------
Substring: AB
Proper prefix: A, Proper suffix: B
------------------------------
Substring: ABC
Proper prefix: A, Proper suffix: C
Proper prefix: AB, Proper suffix: BC
------------------------------
Substring: ABCA
Proper prefix: A, Proper suffix: A *1
Proper prefix: AB, Proper suffix: CA
Proper prefix: ABC, Proper suffix: BCA
------------------------------
Substring: ABCAA
Proper prefix: A, Proper suffix: A *1
Proper prefix: AB, Proper suffix: AA
Proper prefix: ABC, Proper suffix: CAA
Proper prefix: ABCA, Proper suffix: BCAA
------------------------------
Substring: ABCAAB
Proper prefix: A, Proper suffix: B
Proper prefix: AB, Proper suffix: AB *2
Proper prefix: ABC, Proper suffix: AAB
Proper prefix: ABCA, Proper suffix: CAAB
Proper prefix: ABCAA, Proper suffix: BCAAB
------------------------------
Substring: ABCAABC
Proper prefix: A, Proper suffix: C
Proper prefix: AB, Proper suffix: BC
Proper prefix: ABC, Proper suffix: ABC *3
Proper prefix: ABCA, Proper suffix: AABC
Proper prefix: ABCAA, Proper suffix: CAABC
Proper prefix: ABCAAB, Proper suffix: BCAABC
------------------------------
```

Task 3.2: Manually, find the nextArray (lps array):

(a) ABCDE

j	Pattern [0 ..j-1]	Proper prefixes	Proper Suffixes	next[j]
0	-	null	null	-1
1	A	λ	-	0
2	AB	λ, A	B	0
3	ABC	λ, A, AB	C, BC	0
4	ABCD	λ, A, AB, ABC	D, CD, BCD	0
5	ABCDE	λ, A, AB, ABC, ABCD	E, DE, CDE, BCDE	0

The next array is:

(b) AAAAA

j	Pattern [0 ..j-1]	Proper prefixes	Proper Suffixes	next[j]
0	-	null	null	-1
1	A	λ	-	0
2	AA	λ,A	A	1
3	AAA	λ, A,AA	A,AA	2
4	AAAA	λ, A,AA,AAA	A,AA,AAA	3
5	AAAAA	λ, A, AA, AAA, AAAA	A, AA, AAA,AAAA	4

The next array is:

(c) ABABAMK

j	Pattern [0 ..j-1]	Proper prefixes	Proper Suffixes	next[j]
0	-	null	null	-1
1	A	λ	-	0
2	AB	λ, A	B	0
3	ABA	λ, A, AB	A,BA	1
4	ABAB	λ, A, AB, ABA	B, AB, BAB	2
5	ABABA	λ, A, AB, ABA, ABAB	A, BA, ABA, BABA	3
6	ABABAM	λ, A, AB, ABA, ABAB, ABABA	M, AM, BAM, ABAM, BABAM	0
7	ABABAMK	λ, A, AB, ABA, ABAB, ABABAM	K, MK, AMK, BAMK, ABAMK, BABAMK	0

Task 4: Knuth-Morris-Pratt (KMP) Implementation:

We declare M and N, which are the lengths of the pattern and the text, then we Preprocess the pattern (calculate lps[] array), so lps[] will hold the longest prefix suffix values for pattern, we find lps by calling another method called “computeLPSArray(String pattern)” which basically returns a Next[j] pattern, Compare characters at text.charAt(i) and pattern.charAt(j), f they match, increment both i and j, If j reaches the end of the pattern (j == M), a match is found. Add the starting index to the indexes string and update j using the LPS array. Return the indexes string or a message if no match is found.

Code:

```
// task 4
1 usage
public static String searchKMP(String pattern, String text)
{
    int M = pattern.length();
    int N = text.length();
    String indexes = "";

    // Preprocess the pattern (calculate lps[] array)
    // lps[] will hold the longest prefix suffix values for pattern
    int[] lps = computeLPSArray(pattern);

    int i = 0; // index for txt[]
    int j = 0; // index for pat[]
    while (i < N) {
        if (pattern.charAt(j) == text.charAt(i)) {
            j++;
            i++;
        }
        if (j == M) {
            indexes += (i - j) + "  ";
            j = lps[j - 1];
        }

        // mismatch after j matches
        else if (i < N && pattern.charAt(j) != text.charAt(i)) {
            // Do not match lps[0..lps[j-1]] characters, they will match anyway
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
    if (indexes.isEmpty())
        return "Pattern not in text.";
    return indexes;
}
```

```java
1 usage
static int[] computeLPSArray(String pattern) {
    int M = pattern.length();
    int lps[] = new int[M];
    // length of the previous longest prefix suffix
    int len = 0;
    int i = 1;
    lps[0] = 0; // lps[0] is always 0

    // the loop calculates lps[i] for i = 1 to M-1
    while (i < M) {
        if (pattern.charAt(i) == pattern.charAt(len)) {
            len++;
            lps[i] = len;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            if (len != 0) {
                len = lps[len - 1];
            }
            else // if (len == 0)
            {
                lps[i] = len;
                i++;
            }
        }
    }

    return lps;
}
```

Output:

```
Enter a text : ABABCABABABCABABCABABCABABKKKABABCABAB
Enter a pattern to search for: ABABCABAB
Pattern found at these text starting indexes: 0  7  12  17  29
```