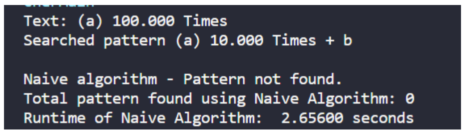
Name : Ramzy Izza Wardhana
Class : IUP CS 1
NIM : 21/472698/PA/20322

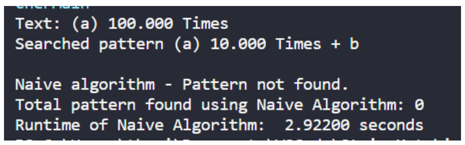
Assignment 10 – Lab Algorithm and Data Structure

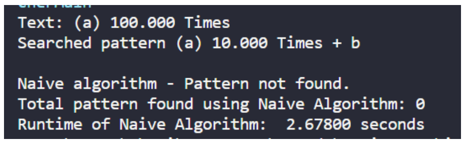
Complete Source Code (Zipped File):
https://drive.google.com/file/d/1NctuNZQqORoznsS4in1q74HnwpJj4FtG/view?usp=sharing

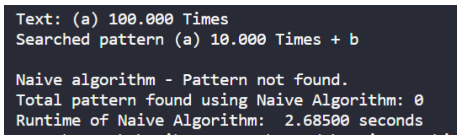
1. Given a text = "aaaa...aa" (100.000 times of letter a) and a pattern = "aaa...aab" (10.000 times of letter a and 1 letter b). Compare the running time of naïve algorithm and KMP algorithm to perform string matching! Perform the string matching 10 times, and then compute the average running time of each algorithm

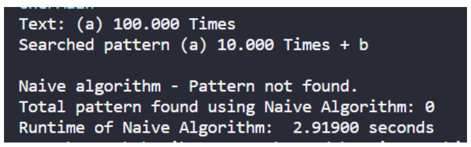
Naïve Algorithm:

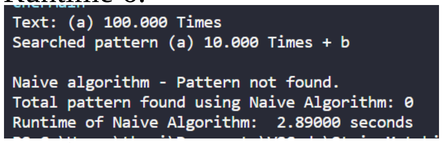
Runtime 1:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.65600 seconds
```

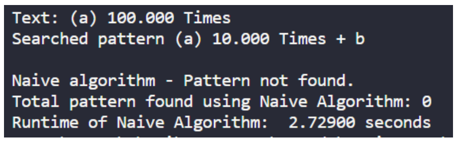
Runtime 2:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.92200 seconds
```

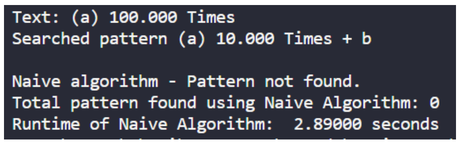
Runtime 3:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.67800 seconds
```

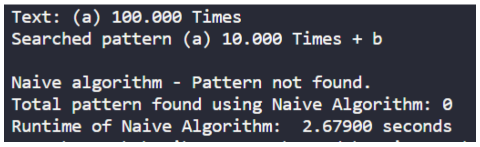
Runtime 4:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.68500 seconds
```

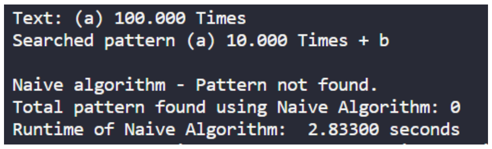
Runtime 5:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.91900 seconds
```

Runtime 6:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.89000 seconds
```

Runtime 7:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.72900 seconds
```

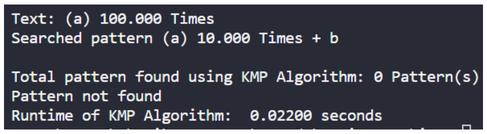
Runtime 8:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.89000 seconds
```

Runtime 9:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.67900 seconds
```

Runtime 10:

```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Naive algorithm - Pattern not found.
Total pattern found using Naive Algorithm: 0
Runtime of Naive Algorithm:  2.83300 seconds
```

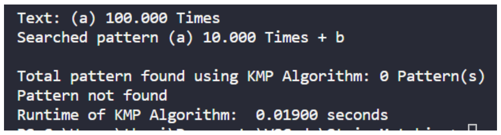
KMP Algorithm:

Runtime 1:



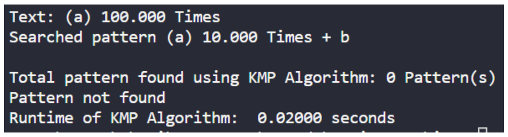
```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.02200 seconds
```

Runtime 2:



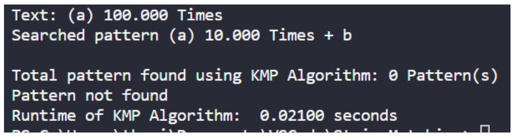
```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.01900 seconds
```

Runtime 3:



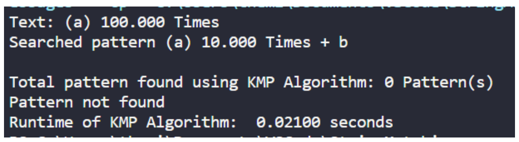
```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.02000 seconds
```

Runtime 4:



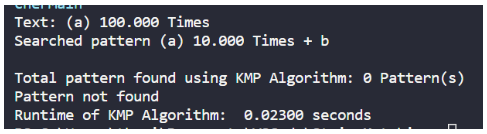
```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.02100 seconds
```

Runtime 5:



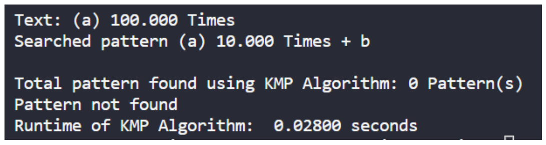
```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.02100 seconds
```

Runtime 6:



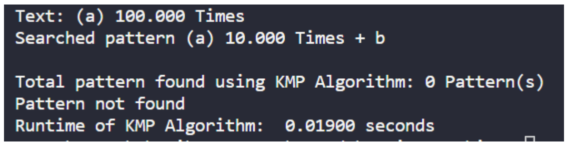
```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.02300 seconds
```

Runtime 7:



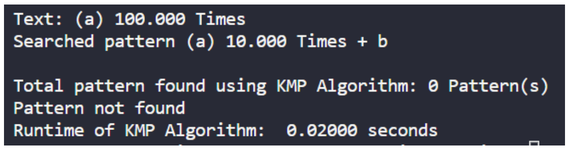
```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.02800 seconds
```

Runtime 8:



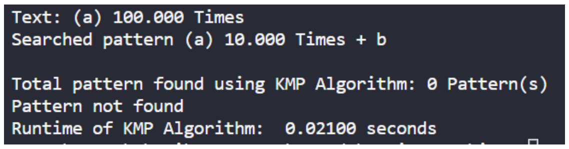
```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.01900 seconds
```

Runtime 9:



```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.02000 seconds
```

Runtime 10:



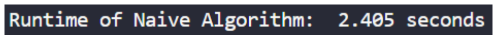
```
Text: (a) 100.000 Times
Searched pattern (a) 10.000 Times + b

Total pattern found using KMP Algorithm: 0 Pattern(s)
Pattern not found
Runtime of KMP Algorithm:  0.02100 seconds
```

Problem Number 1											
Runtime in Seconds	Runtime 1	Runtime 2	Runtime 3	Runtime 4	Runtime 5	Runtime 6	Runtime 7	Runtime 8	Runtime 9	Runtime 10	**Average (s)**
Naïve Algorithm	2.656	2.922	2.678	2.685	2.919	2.89	2.729	2.89	2.679	2.833	**2.7881**
KMP Algorithm	0.022	0.019	0.02	0.021	0.021	0.023	0.028	0.019	0.02	0.021	**0.0214**

2. Given a text = "aaaa...aa" (100.000 times of letter a) and a pattern = "aaa...aa" (10.000 times of letter a). Compare the running time of naïve algorithm and KMP algorithm (without any output) to perform string matching! Perform the string matching 10 times, and then compute the average running time of each algorithm:

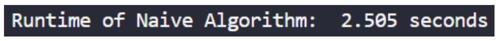
Naïve Algorithm:

Runtime 1:



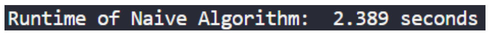
```
Runtime of Naive Algorithm:  2.405 seconds
```

Runtime 2:



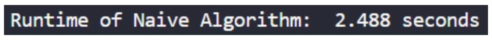
```
Runtime of Naive Algorithm:  2.505 seconds
```

Runtime 3:



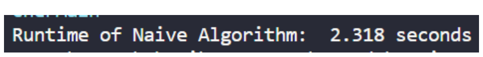
```
Runtime of Naive Algorithm:  2.389 seconds
```

Runtime 4:



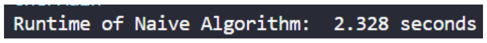
```
Runtime of Naive Algorithm:  2.488 seconds
```

Runtime 5:



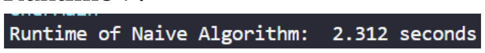
```
Runtime of Naive Algorithm:  2.318 seconds
```

Runtime 6:



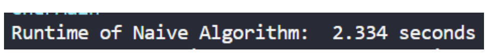
```
Runtime of Naive Algorithm:  2.328 seconds
```

Runtime 7:



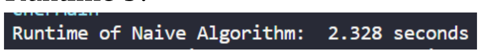
```
Runtime of Naive Algorithm:  2.312 seconds
```

Runtime 8:



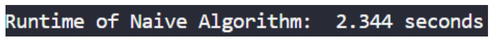
```
Runtime of Naive Algorithm:  2.334 seconds
```

Runtime 9:



```
Runtime of Naive Algorithm:  2.328 seconds
```

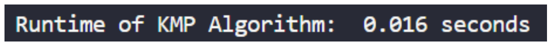
Runtime 10:



```
Runtime of Naive Algorithm:  2.344 seconds
```

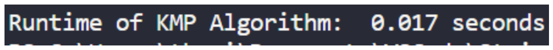
KMP Algorithm:

Runtime 1:



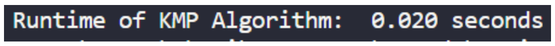
```
Runtime of KMP Algorithm:  0.016 seconds
```

Runtime 2:



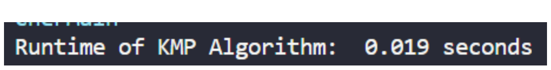
```
Runtime of KMP Algorithm:  0.017 seconds
```

Runtime 3:



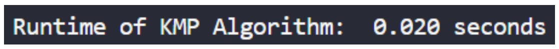
```
Runtime of KMP Algorithm:  0.020 seconds
```

Runtime 4:



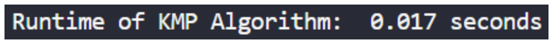
```
Runtime of KMP Algorithm:  0.019 seconds
```

Runtime 5:



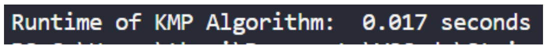
```
Runtime of KMP Algorithm:  0.020 seconds
```

Runtime 6:



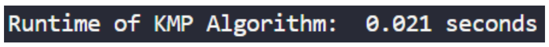
```
Runtime of KMP Algorithm:  0.017 seconds
```

Runtime 7:



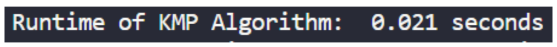
```
Runtime of KMP Algorithm:  0.017 seconds
```

Runtime 8:



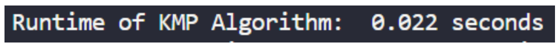
```
Runtime of KMP Algorithm:  0.021 seconds
```

Runtime 9:



```
Runtime of KMP Algorithm:  0.021 seconds
```

Runtime 10:

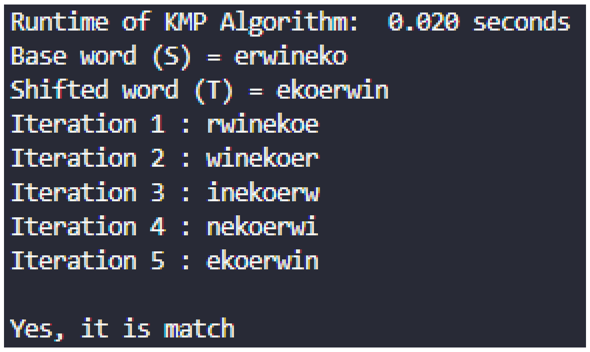


```
Runtime of KMP Algorithm:  0.022 seconds
```

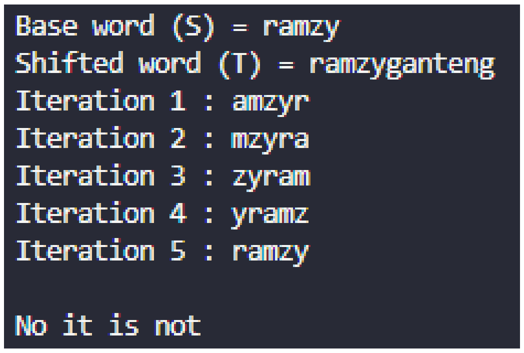
Problem Number 2											
Runtime in Seconds	Runtime 1	Runtime 2	Runtime 3	Runtime 4	Runtime 5	Runtime 6	Runtime 7	Runtime 8	Runtime 9	Runtime 10	**Average (s)**
Naïve Algorithm	2.405	2.505	2.389	2.488	2.318	2.328	2.312	2.334	2.328	2.334	**2.3741**
KMP Algorithm	0.016	0.017	0.02	0.019	0.02	0.017	0.017	0.021	0.021	0.022	**0.019**

3. Given two words S and T, both has equal length (the maximum length is 100.000). Your task is to determine whether T can be created by performing multiple cycle shifts to S (a cycle shift is a transfer of the first character of the string to the end of this string). For example, if S = "erwineko" and T = "ekoerwin", the answer must be "YES", because "erwineko" -> "rwinekoe" -> "winekoer" -> "inekoerw" -> "nekoerwi" -> "ekoerwin". In this problem, you have to use the KMP algorithm to solve this task (the approach may not be obvious, but you have to figure the usage of the KMP algorithm in this problem)

   Output:



```
Runtime of KMP Algorithm:  0.020 seconds
Base word (S) = erwineko
Shifted word (T) = ekoerwin
Iteration 1 : rwinekoe
Iteration 2 : winekoer
Iteration 3 : inekoerw
Iteration 4 : nekoerwi
Iteration 5 : ekoerwin

Yes, it is match
```



```
Base word (S) = ramzy
Shifted word (T) = ramzyganteng
Iteration 1 : amzyr
Iteration 2 : mzyra
Iteration 3 : zyram
Iteration 4 : yramz
Iteration 5 : ramzy

No it is not
```

Main.java

```java
package StringMatching;

import java.text.DecimalFormat;
import java.text.NumberFormat;

public class StringMatcherMain {
    public static void main(String[] args){

        long timerStart = System.currentTimeMillis(); //start execution time in
miliseconds

        String text = "a";//multiply a by 100.000 times for text
        String textRepeated = text.repeat(100000);

        String pattern = "a";//multiply a by 10.000 times for pattern
        String patternRepeated = pattern.repeat(10000) + "b";
        String patternRepeated2 = pattern.repeat(10000);

        // System.out.println("Text: (a) 100.000 Times\nSearched pattern (a)
10.000 Times + b\n");
        // System.out.println("Text: (a) 100.000 Times\nSearched pattern (a)
10.000 Times\n");

        // StringMatching.naive(textRepeated, patternRepeated2); //naive
algorithm
        StringMatching.kmp(textRepeated, patternRepeated2); //KMP algorithm

        long endTimer = System.currentTimeMillis(); //end execution time in
millisecondw

        NumberFormat formatter = new DecimalFormat("#0.000"); //convert
miliseconds to seconds
        System.out.print("Runtime of KMP Algorithm:  " +
formatter.format((endTimer - timerStart) / 1000d) + " seconds");
        // System.out.print("Runtime of KMP Algorithm:  " +
formatter.format((endTimer - timerStart) / 1000d) + " seconds");

        // StringMatching.cycleShift("erwineko", "ekoerwin");

        StringMatching.cycleShift("ramzy", "ramzyganteng");

    }
}
```

StringMatching.java

```java
package StringMatching;

public class StringMatching {

    static void naive(String text, String pattern){
        int textLength = text.length();
        int patternLength = pattern.length();
        int naiveCounter = 0;
        boolean found = false;

        for(int i = 0; i +  patternLength <= textLength; i ++){
            boolean currentFound = true;
            for(int j = 0; j < patternLength; j++){
                if(text.charAt(i+j) != pattern.charAt(j)){
                    currentFound = false;
                    break;
                }
            }
            if(currentFound){
                found = true;
                // naiveCounter++;
            }
        }
        // if(!found){
        //     System.out.println("Naive algorithm - Pattern not found.");
        // }
        // System.out.println("Total pattern found using Naive Algorithm: " +
naiveCounter);
    }

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

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

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

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

    static void kmp(String text, String pattern){
        String combined = pattern + "#" + text;
        int combineLength = combined.length();
        int patternLength = pattern.length();
        int KMPCounter = 0;

        int lps[] = computeLPSArray(combined);
        boolean found = false;

        for(int i = patternLength + 1; i < combineLength; i++){
            if(lps[i] == patternLength){
                found = true;
                // KMPCounter++;
            }
        }
        // System.out.println("Total pattern found using KMP Algorithm: " +
KMPCounter + " Pattern(s)");

        // if(!found){
        //     System.out.println("Pattern not found");
        // }
    }

    static boolean kmpFindIndex(String text, String pattern){
        String combined = pattern + "#" + text;
        int combineLength = combined.length();
        int patternLength = pattern.length();

        int lps[] = computeLPSArray(combined);
        boolean found = false;
        for(int i = patternLength + 1; i < combineLength; i++){
            if(lps[i] == patternLength)
                found = true;
        }
        if(!found)
            found = false;
        return found;
    }


    static String shiftChar(String text, char ch, int i) {
        StringBuilder s = new StringBuilder(text);
        s.setCharAt(i, ch);
        return s.toString();
    }

    static void cycleShift(String S, String T){
        int shiftCounter = 0; int iteration  = 1; int length = S.length();
        boolean index = kmpFindIndex(S,T); char shift;
        System.out.println("\nBase word (S) = " + S + "\nShifted word (T) = " +
T);
        while(shiftCounter < length && !index){
            int i = 0;
            shift = S.charAt(0);

            while(i != length - 1){
                S = shiftChar(S, S.charAt(i + 1), i); i++;
            }

            S = shiftChar(S, shift, length - 1);
            System.out.println("Iteration " + iteration + " : " + S);
            index = kmpFindIndex(S,T);
            shiftCounter++; iteration++;
        }
        String result = (index) ? "\nYes, it is match" : "\nNo it is not";
        System.out.println(result);
    }

        // static void cycleShift(String s, String t){
    //     StringBuilder S = new StringBuilder(s);
    //  StringBuilder T = new StringBuilder(t);
    //     int len = S.length();
    //  int cycleShifts = 0, count = 0;

    //     System.out.println("\nBase word (S) = " + S);
    //     System.out.println("Shifted word (T) = "+ T);
    //     for(; cycleShifts < len; cycleShifts++){
    //         if(S.toString().equals(T.toString())){
    //             System.out.println("Yes, It's Match!");
    //             break;
    //         }
    //      char firstChar = S.charAt(0);
    //         S.substring(1);
    //      S.deleteCharAt(0);
    //      S.append(firstChar);
    //         count++;
    //      System.out.println("Iteration " + count + " : " +  S.toString());

    //         if(cycleShifts == (len-1) && !S.toString().equals(T.toString())){
    //             System.out.println("No, It is not Match");
    //         }
    //  }
    // }
}
```