

Day 11:

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.

```
public class StringOperations {  
  
    public static String middleSubstringAfterConcatAndReverse(String str1,  
String str2, int length) {  
        if (str1 == null || str2 == null || length <= 0) {  
            return "";  
        }  
  
        String concatenated = str1.concat(str2);  
  
        StringBuilder reversed = new StringBuilder(concatenated).reverse();  
  
        int startIndex = (reversed.length() - length) / 2;  
        int endIndex = startIndex + length;  
        if (startIndex < 0 || endIndex > reversed.length()) {  
            return "";  
        }  
  
        return reversed.substring(startIndex, endIndex);  
    }  
}
```

```

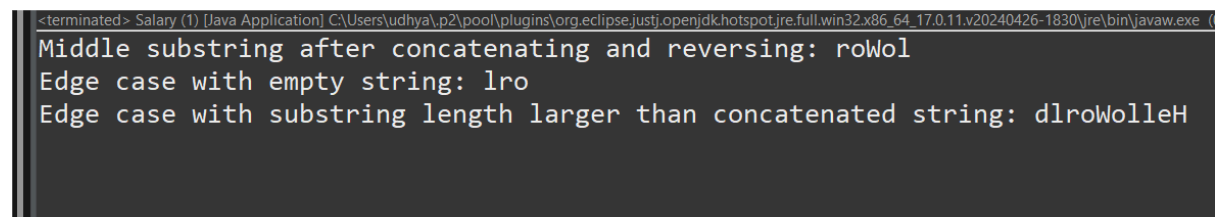
    }

    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "World";
        int length = 5;
        System.out.println("Middle substring after concatenating and reversing: "
+
            middleSubstringAfterConcatAndReverse(str1, str2, length));

        System.out.println("Edge case with empty string: " +
            middleSubstringAfterConcatAndReverse("", "World", 3));
        System.out.println("Edge case with substring length larger than concatenated
string: " +
            middleSubstringAfterConcatAndReverse("Hello", "World", 10));    }
    }

```

OUTPUT:



```

<terminated> Salary (1) [Java Application] C:\Users\udhya\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe (
Middle substring after concatenating and reversing: roWo1
Edge case with empty string: lro
Edge case with substring length larger than concatenated string: dlroWolleH

```

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.

```
public class NaivePatternSearch {

    public static void search(String text, String pattern) {
        int n = text.length();
        int m = pattern.length();
        int comparisons = 0;

        for (int i = 0; i <= n - m; i++) {
            int j;
            for (j = 0; j < m; j++) {
                comparisons++;
                if (text.charAt(i + j) != pattern.charAt(j))
                    break;
            }
            if (j == m) {
                System.out.println("Pattern found at index " + i);
            }
        }

        System.out.println("Number of comparisons made: " + comparisons);
    }

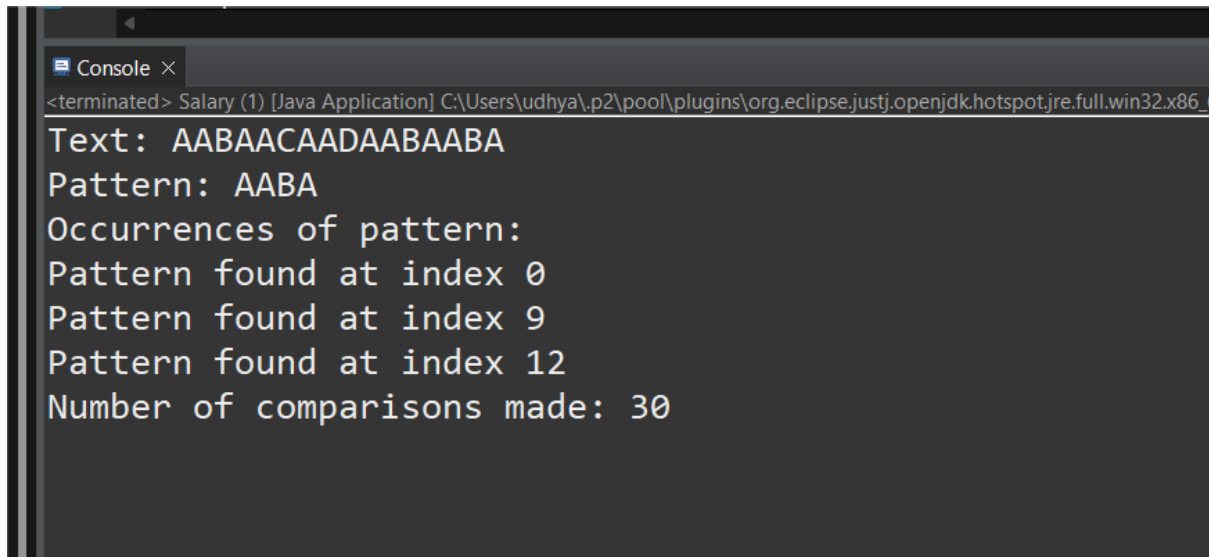
    public static void main(String[] args) {
        String text = "AABAACAADAABAABA";
        String pattern = "AABA";
        System.out.println("Text: " + text);
    }
}
```

```

        System.out.println("Pattern: " + pattern);
        System.out.println("Occurrences of pattern:");
        search(text, pattern);
    }

```

OUTPUT:



```

<terminated> Salary (1) [Java Application] C:\Users\udhya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_
Text: AABAACAADAABAABA
Pattern: AABA
Occurrences of pattern:
Pattern found at index 0
Pattern found at index 9
Pattern found at index 12
Number of comparisons made: 30

```

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.wipro;
```

```
public class Salary {
```

```

    private static void computeLPSArray(String pattern, int m, int[] lps) {
        int length = 0;

```

```
int i = 1;  
lps[0] = 0;
```

```
while (i < m) {  
    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++;  
        }  
    }  
}  
}
```

```
public static void KMPSearch(String pattern, String text) {  
    int m = pattern.length();  
    int n = text.length();
```

```

int[] lps = new int[m];
computeLPSArray(pattern, m, lps);

int i = 0;
int j = 0;

while (i < n) {
    if (pattern.charAt(j) == text.charAt(i)) {
        j++;
        i++;
    }

    if (j == m) {
        System.out.println("Found pattern at index " + (i - j));
        j = lps[j - 1];
    } else if (i < n && pattern.charAt(j) != text.charAt(i)) {
        if (j != 0) {
            j = lps[j - 1];
        } else {
            i++;
        }
    }
}
}

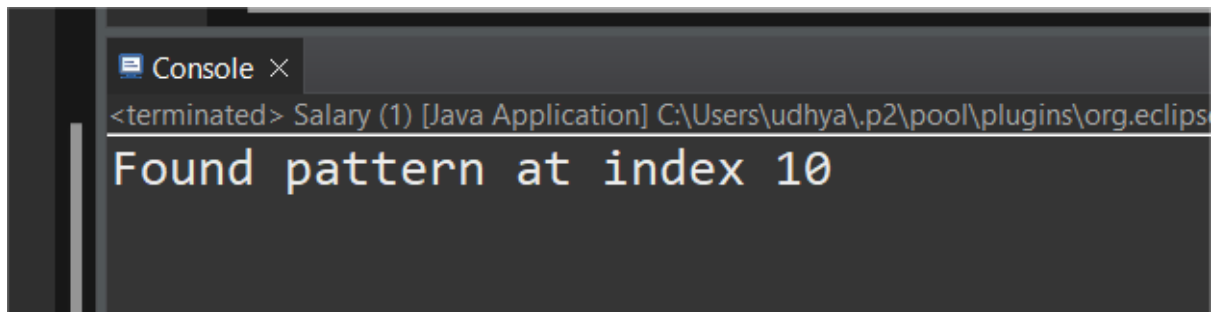
```

```

public static void main(String[] args) {
    String text = "ABABDABACDABABCABAB";
    String pattern = "ABABCABAB";
    KMPSearch(pattern, text);
}
}

```

OUTPUT:



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.

```

public class RabinKarp {

    public static void search(String pattern, String text, int prime) {
        int m = pattern.length();
        int n = text.length();
        int patternHash = 0;
        int textHash = 0;
        int h = 1;
    }
}

```

```

int d = 256;
    for (int i = 0; i < m - 1; i++) {
        h = (h * d) % prime;
    }

for (int i = 0; i < m; i++) {
    patternHash = (d * patternHash + pattern.charAt(i)) % prime;
    textHash = (d * textHash + text.charAt(i)) % prime;
}

for (int i = 0; i <= n - m; i++) {
    if (patternHash == textHash) {

        int j;
        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);
        }
    }
}

```



```

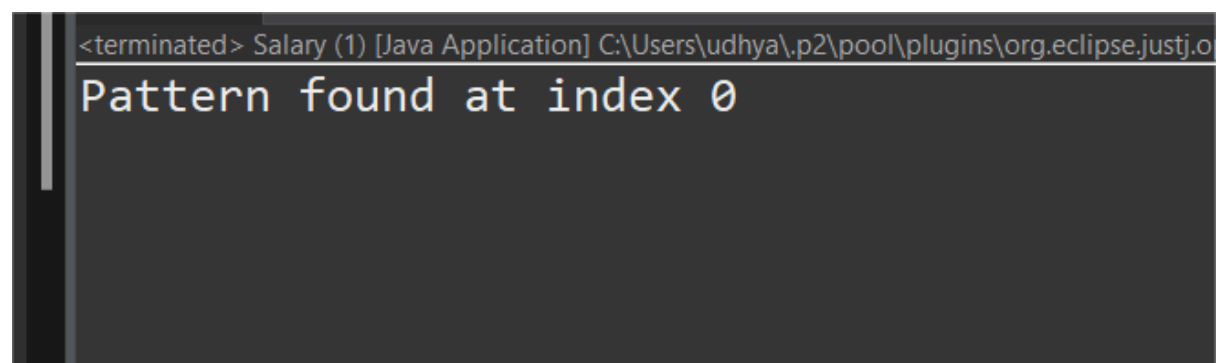
        if (i < n - m) {
            textHash = (d * (textHash - text.charAt(i) * h) + text.charAt(i + 1)) %
prime;

            if (textHash < 0) {
                textHash = (textHash + prime);
            }
        }
    }
}

public static void main(String[] args) {
    String text = "GEEKS FOR GEEKS";
    String pattern = "GEEK";
    int prime = 101;
    search(pattern, text, prime);
}
}

```

OUTPUT:



The screenshot shows a Java application window titled "<terminated> Salary (1) [Java Application] C:\Users\udhya\.p2\pool\plugins\org.eclipse.justj.o". The main content area of the window displays the text "Pattern found at index 0" in a monospaced font.

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.wipro.patterns;

import java.util.Arrays;

public class BoyerMooreAlgorithm {

    private static int NO_OF_CHARS = 256;

    public static void search(String text, String pattern) {
        int m = pattern.length();
        int n = text.length();

        int[] badChar = new int[NO_OF_CHARS];
        badCharHeuristic(pattern, m, badChar);

        int s = 0;
        while (s <= (n - m)) {
            int j = m - 1;

            while (j >= 0 && pattern.charAt(j) == text.charAt(s + j))
                j--;
```

```

        if (j < 0) {
            System.out.println("Pattern found at index " + s);
            s += (s + m < n) ? m - badChar[text.charAt(s + m)] : 1;
        } else {
            s += Math.max(1, j - badChar[text.charAt(s + j)]);
        }
    }
}

private static void badCharHeuristic(String str, int size, int[] badChar) {
    Arrays.fill(badChar, -1);
    for (int i = 0; i < size; i++) {
        badChar[(int) str.charAt(i)] = i;
    }
}

public static void main(String[] args) {
    String text = "ABAAABCD";
    String pattern = "ABC";
    search(text, pattern);
}
}

```

OUTPUT:

Console ×

<terminated> BoyerMooreAlgorithm [Java Application] C:\Users\udhya\.p2\p

Pattern found at index 4