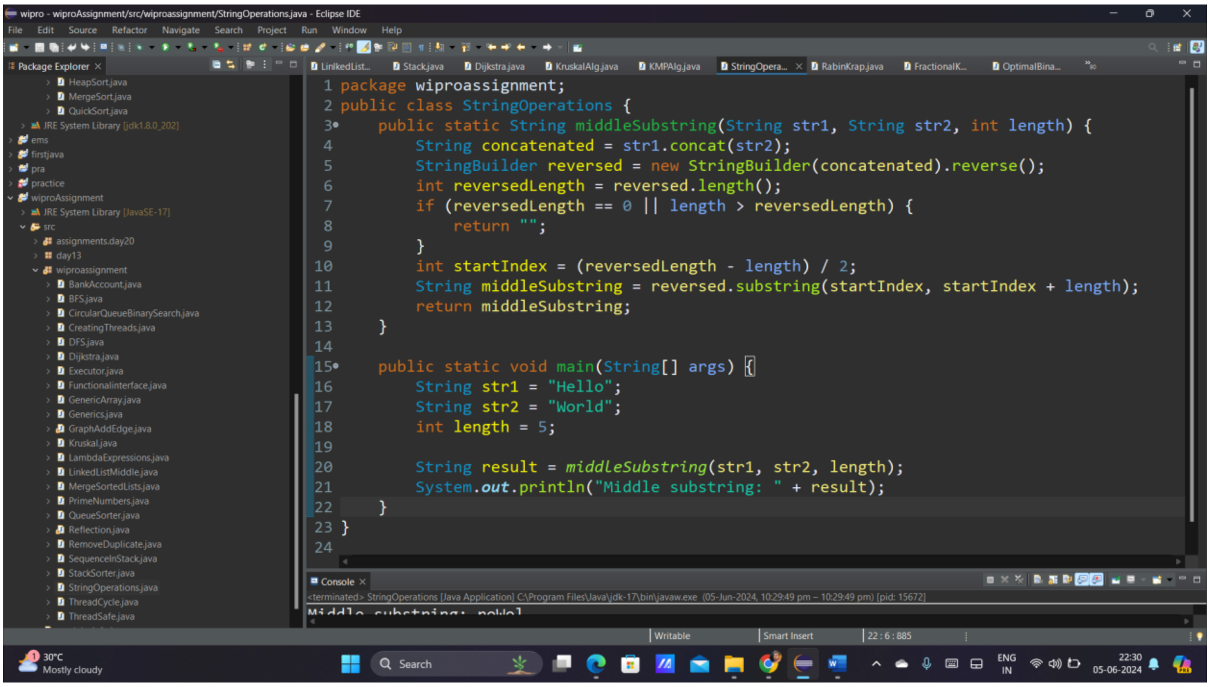
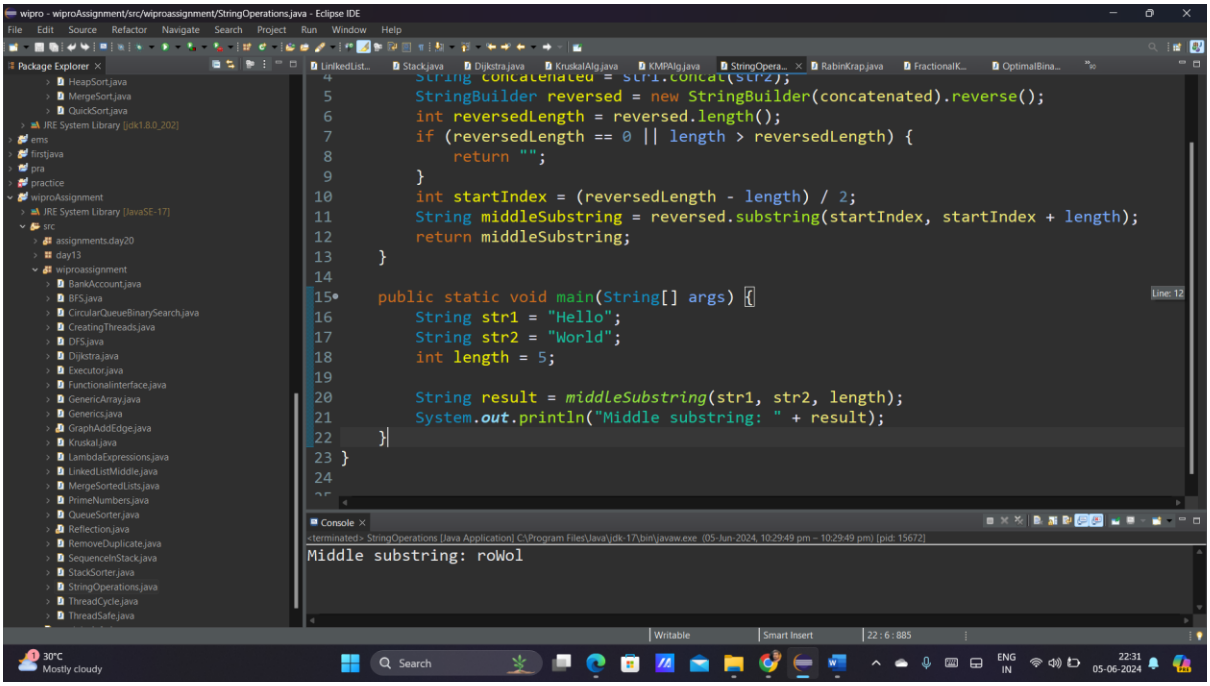
Assignment-Day 11

Task 1: String Operations

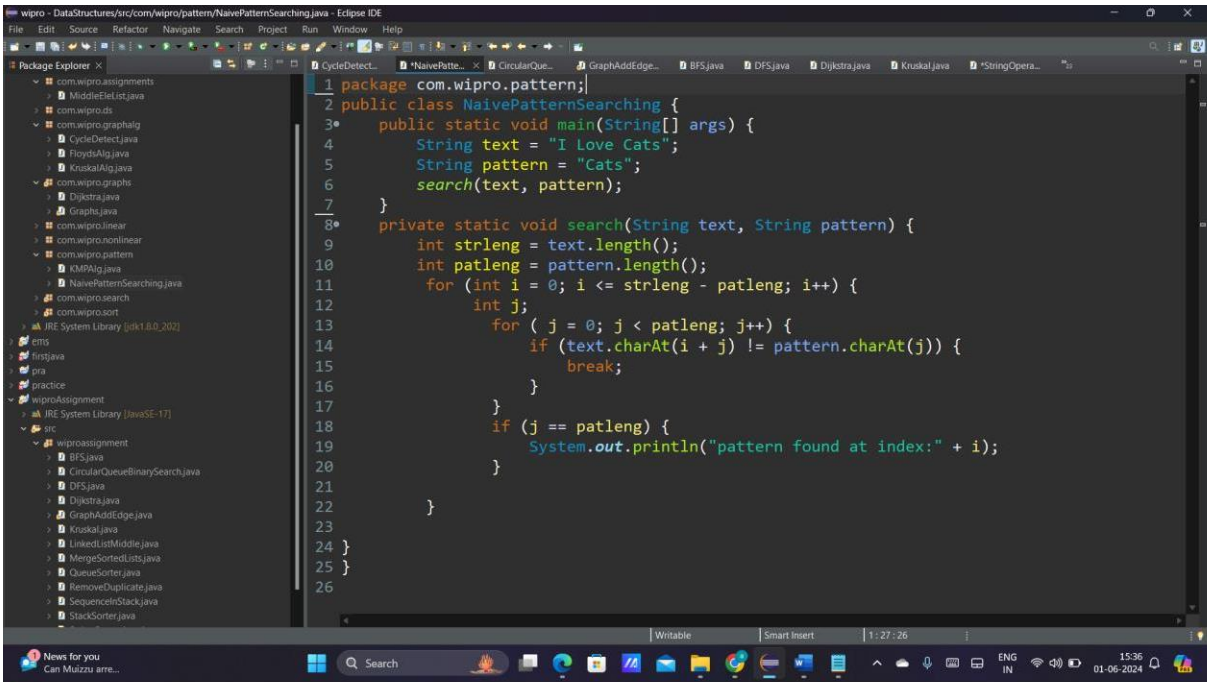
Write a method that takes two strings, concatenates them, reverses the result, and then extracts themiddle substring of the given length. Ensure your method handles edge cases, such as an empty
string or a substring length larger than the concatenated string.

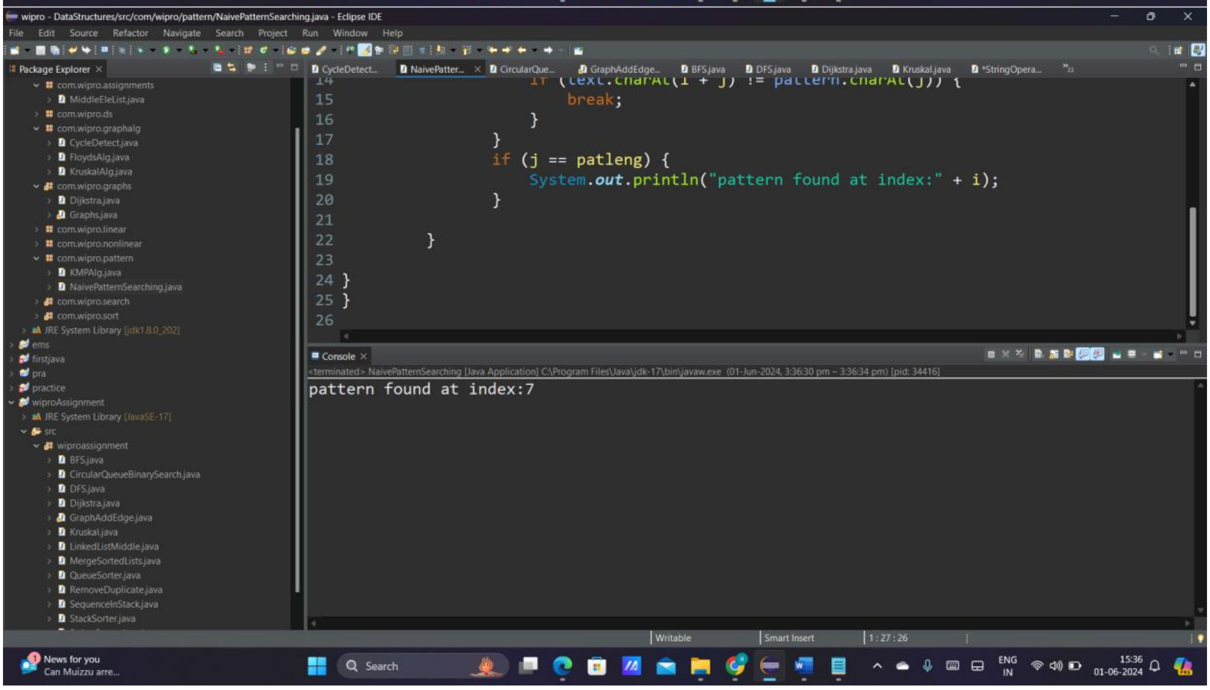




Task 2: Naive Pattern Search

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

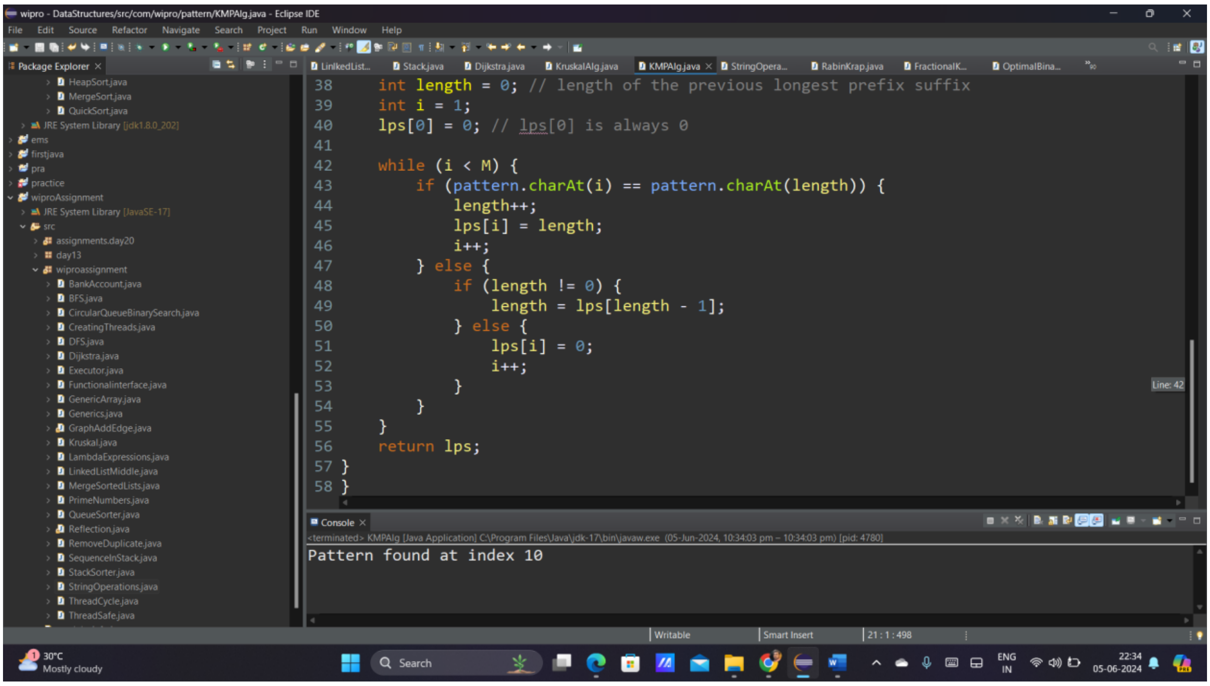




Task 3: Implementing the KMP Algorithm

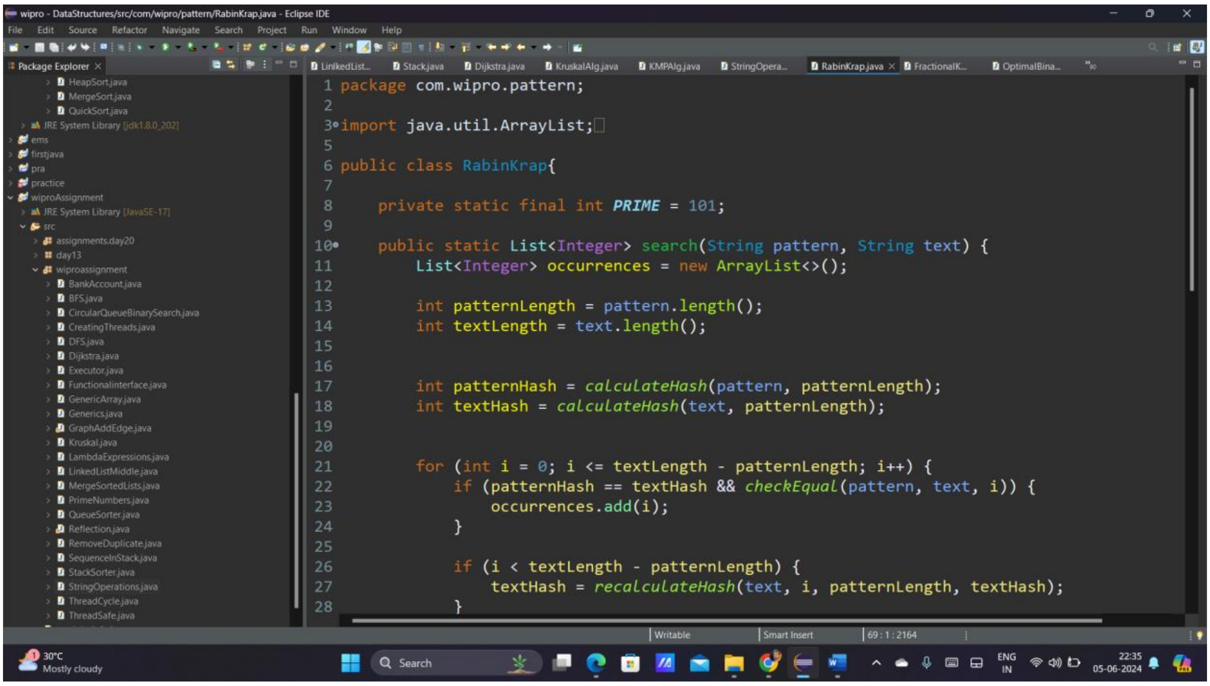
Code the Knuth-Morris-Pratt (KMP) algorithm in C# for pattern searching which pre-processes the pattern to reduce the number of comparisons. Explain how this pre-processing improves the search time compared to the naive approach.

```java
package com.wipro.pattern;
public class KMPAlg{
public static void main(String[] args) {
    String text = "ABABDABACDABABCABAB";
    String pattern = "ABABCABAB";
    search(text, pattern);
}

public static void search(String text, String pattern) {
    int[] lps = computeLPSArray(pattern);
    int i = 0; // index for text
    int j = 0; // index for pattern
    int M = pattern.length();
    int N = text.length();

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

        if (j == M) {
            System.out.println("Pattern found 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++;
            }
        }
    }
}

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



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.

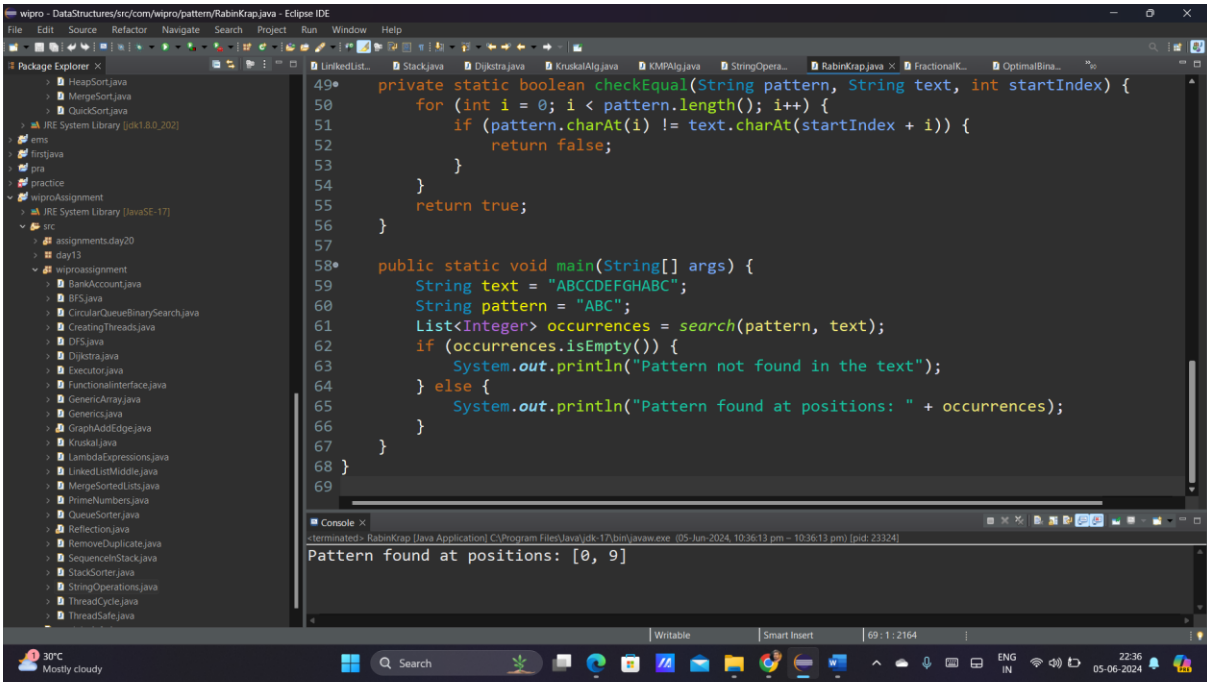


wipro - DataStructures/src/com/wipro/pattern/RabinKrap.java - Eclipse IDE

```java
            textHash = recalculateHash(text, i, patternLength, textHash);
            }
        }

        return occurrences;
    }

    private static int calculateHash(String str, int length) {
        int hash = 0;
        for (int i = 0; i < length; i++) {
            hash += str.charAt(i) * Math.pow(PRIME, i);
        }
        return hash;
    }

    private static int recalculateHash(String str, int oldIndex, int patternLength, int ol
        int newHash = oldHash - str.charAt(oldIndex);
        newHash /= PRIME;
        newHash += str.charAt(oldIndex + patternLength) * Math.pow(PRIME, patternLength -
        return newHash;
    }

    private static boolean checkEqual(String pattern, String text, int startIndex) {
        for (int i = 0; i < pattern.length(); i++) {
            if (pattern.charAt(i) != text.charAt(startIndex + i)) {
                return false;
            }
```

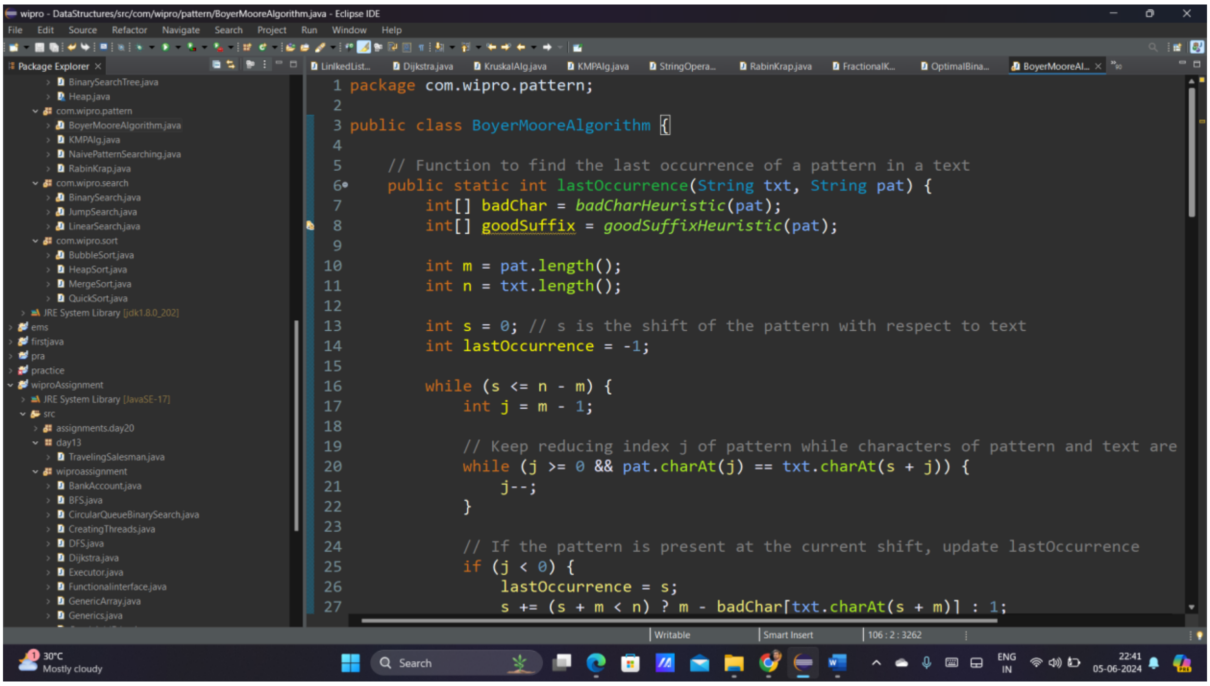
wipro - DataStructures/src/com/wipro/pattern/RabinKrap.java - Eclipse IDE

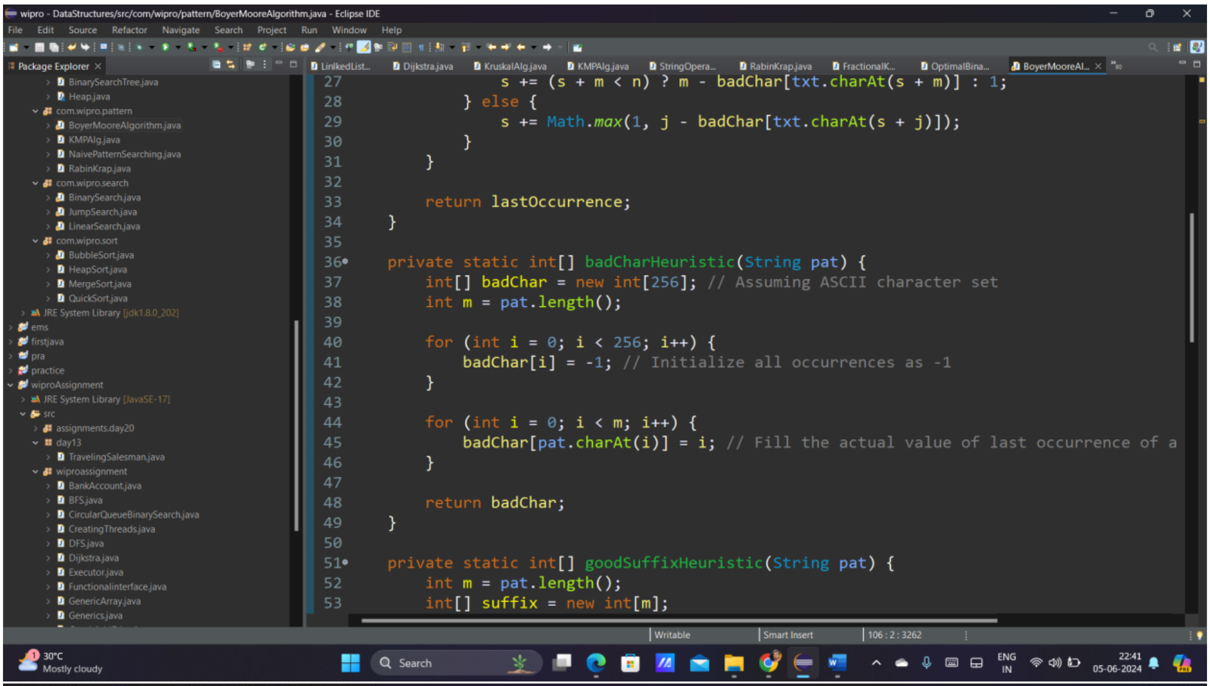
```java
        int newHash = oldHash - str.charAt(oldIndex);
        newHash /= PRIME;
        newHash += str.charAt(oldIndex + patternLength) * Math.pow(PRIME, patternLength -
        return newHash;
    }

    private static boolean checkEqual(String pattern, String text, int startIndex) {
        for (int i = 0; i < pattern.length(); i++) {
            if (pattern.charAt(i) != text.charAt(startIndex + i)) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        String text = "ABCCDEFGHABC";
        String pattern = "ABC";
        List<Integer> occurrences = search(pattern, text);
        if (occurrences.isEmpty()) {
            System.out.println("Pattern not found in the text");
        } else {
            System.out.println("Pattern found at positions: " + occurrences);
        }
    }
}
```

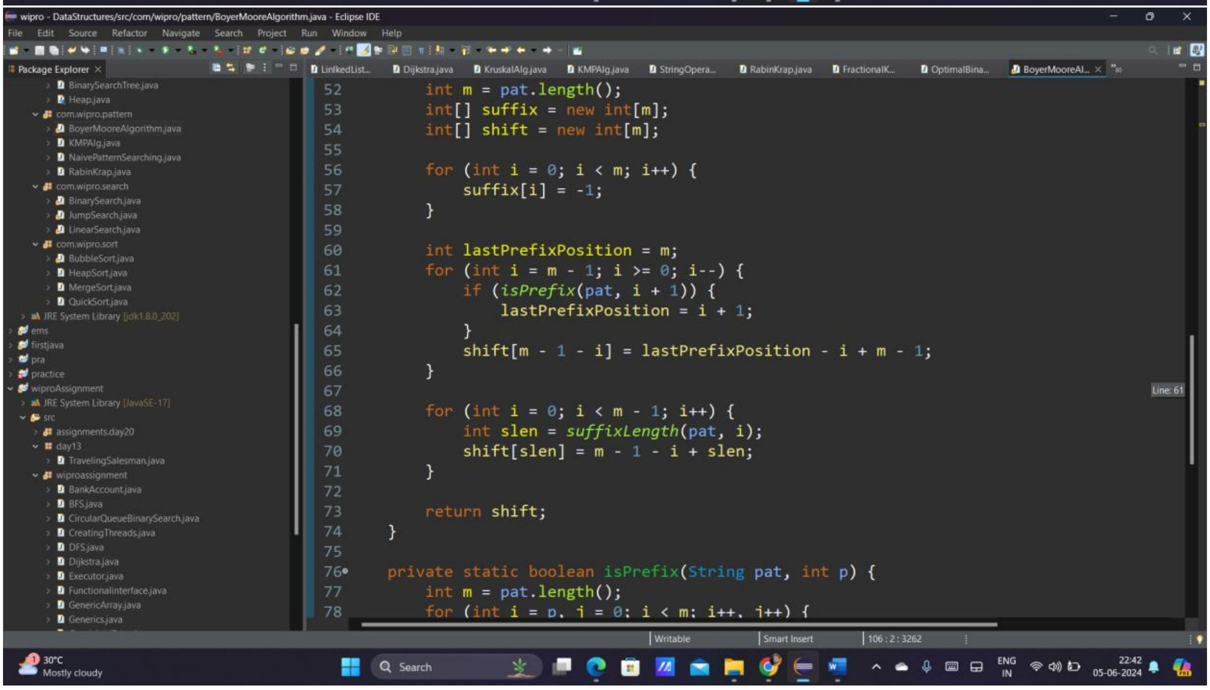


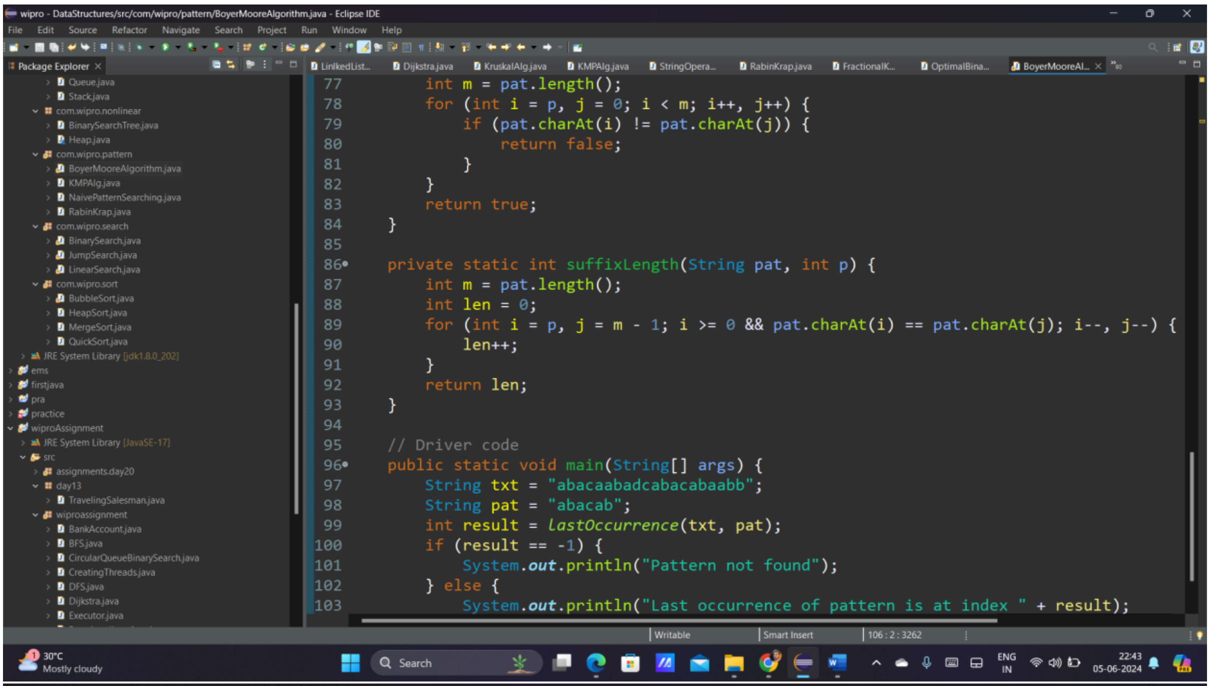
Task 5: Boyer-Moore Algorithm Application

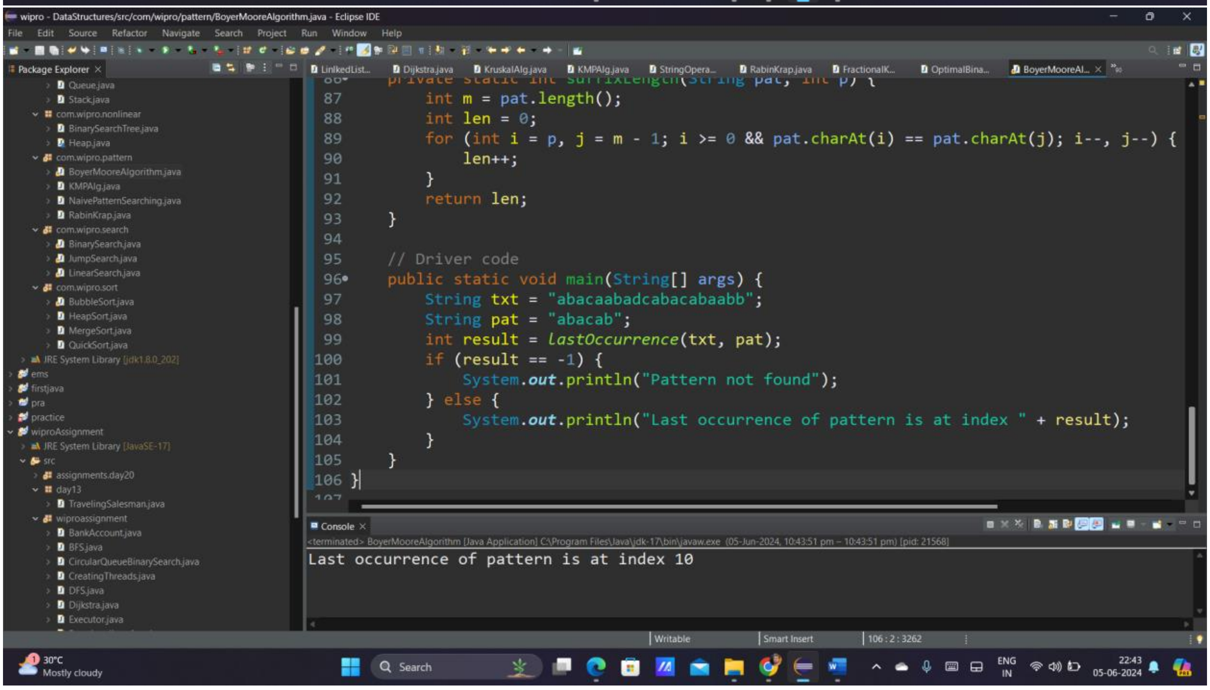
Use the Boyer-Moore algorithm to write a function that finds the last occurrence of a substring in a given string and returns its index. Explain why this algorithm can outperform others in certain scenarios.




wipro - DataStructures/src/com/wipro/pattern/BoyerMooreAlgorithm.java - Eclipse IDE
s += (s + m < n) ? m - badChar[txt.charAt(s + m)] : 1;
} else {
s += Math.max(1, j - badChar[txt.charAt(s + j)]);
}
}
return lastOccurrence;
}
private static int[] badCharHeuristic(String pat) {
int[] badChar = new int[256]; // Assuming ASCII character set
int m = pat.length();
for (int i = 0; i < 256; i++) {
badChar[i] = -1; // Initialize all occurrences as -1
}
for (int i = 0; i < m; i++) {
badChar[pat.charAt(i)] = i; // Fill the actual value of last occurrence of a
}
return badChar;
}
private static int[] goodSuffixHeuristic(String pat) {
int m = pat.length();
int[] suffix = new int[m];


wipro - DataStructures/src/com/wipro/pattern/BoyerMooreAlgorithm.java - Eclipse IDE
int m = pat.length();
int[] suffix = new int[m];
int[] shift = new int[m];
for (int i = 0; i < m; i++) {
suffix[i] = -1;
}
int lastPrefixPosition = m;
for (int i = m - 1; i >= 0; i--) {
if (isPrefix(pat, i + 1)) {
lastPrefixPosition = i + 1;
}
shift[m - 1 - i] = lastPrefixPosition - i + m - 1;
}
for (int i = 0; i < m - 1; i++) {
int slen = suffixLength(pat, i);
shift[slen] = m - 1 - i + slen;
}
return shift;
}
private static boolean isPrefix(String pat, int p) {
int m = pat.length();
for (int i = p, j = 0; i < m; i++, j++) {


wipro - DataStructures/src/com/wipro/pattern/BoyerMooreAlgorithm.java - Eclipse IDE
int m = pat.length();
for (int i = p, j = 0; i < m; i++, j++) {
if (pat.charAt(i) != pat.charAt(j)) {
return false;
}
}
return true;
}
private static int suffixLength(String pat, int p) {
int m = pat.length();
int len = 0;
for (int i = p, j = m - 1; i >= 0 && pat.charAt(i) == pat.charAt(j); i--, j--) {
len++;
}
return len;
}
// Driver code
public static void main(String[] args) {
String txt = "abacaabadcabacabaabb";
String pat = "abacab";
int result = lastOccurrence(txt, pat);
if (result == -1) {
System.out.println("Pattern not found");
} else {
System.out.println("Last occurrence of pattern is at index " + result);


wipro - DataStructures/src/com/wipro/pattern/BoyerMooreAlgorithm.java - Eclipse IDE
int m = pat.length();
int len = 0;
for (int i = p, j = m - 1; i >= 0 && pat.charAt(i) == pat.charAt(j); i--, j--) {
len++;
}
return len;
}
// Driver code
public static void main(String[] args) {
String txt = "abacaabadcabacabaabb";
String pat = "abacab";
int result = lastOccurrence(txt, pat);
if (result == -1) {
System.out.println("Pattern not found");
} else {
System.out.println("Last occurrence of pattern is at index " + result);
}
}
}
Console
Last occurrence of pattern is at index 10