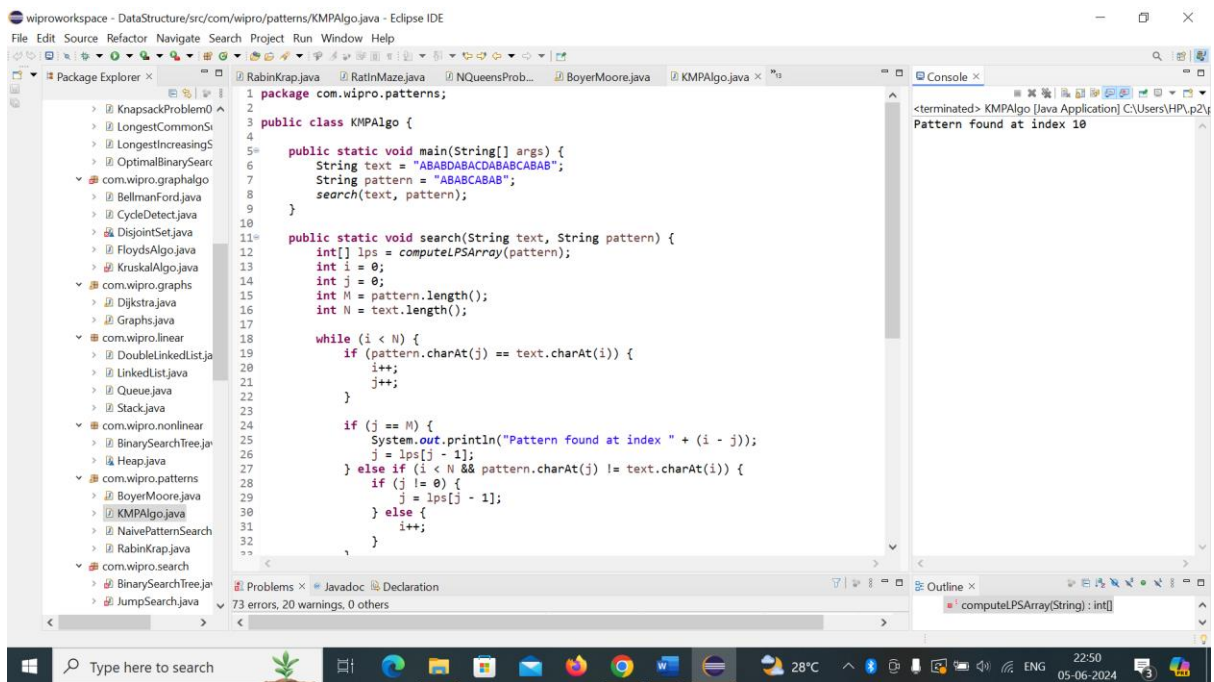


# Day – 11

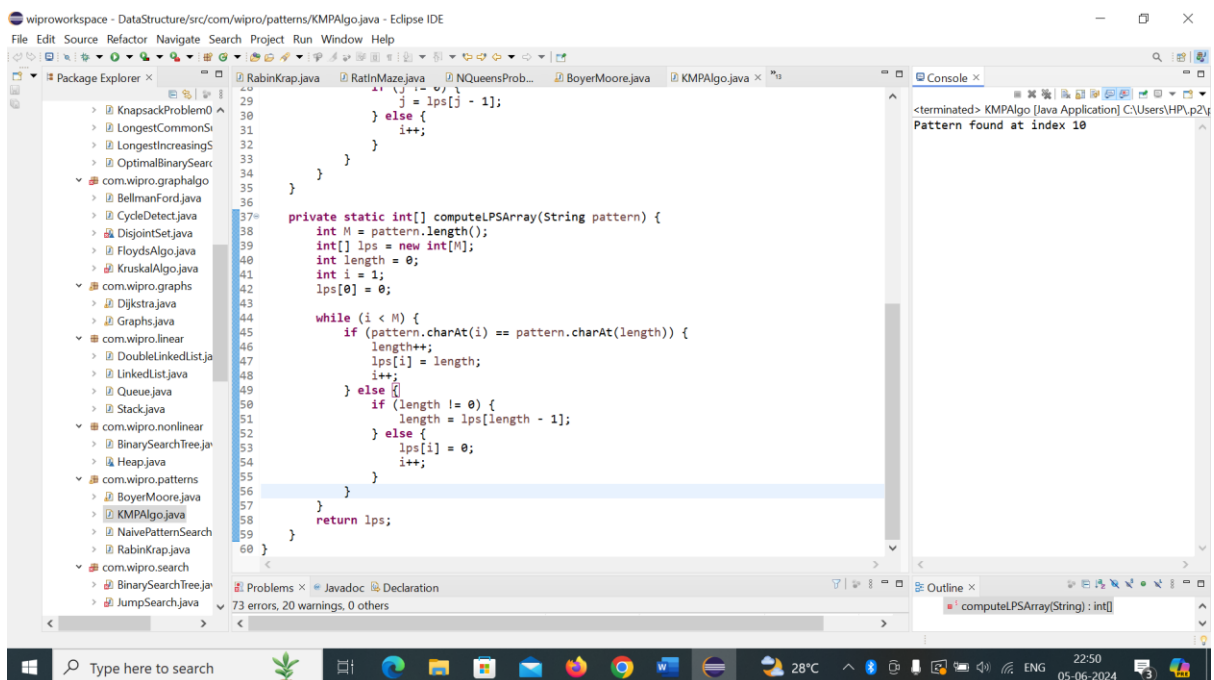
## Task 3: Implementing the KMP Algorithm

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.



The screenshot shows the Eclipse IDE with the KMPAlgo.java file open. The code defines a package `com.wipro.patterns` and a class `KMPAlgo`. The `main` method takes command-line arguments for text and pattern, and calls the `search` method. The `search` method uses the `computeLPSArray` method to pre-process the pattern and then searches for the pattern in the text.

```
1 package com.wipro.patterns;
2
3 public class KMPAlgo {
4
5     public static void main(String[] args) {
6         String text = "ABABDABACDABABCABAB";
7         String pattern = "ABABCABAB";
8         search(text, pattern);
9     }
10
11     public static void search(String text, String pattern) {
12         int[] lps = computeLPSArray(pattern);
13         int i = 0;
14         int j = 0;
15         int M = pattern.length();
16         int N = text.length();
17
18         while (i < N) {
19             if (pattern.charAt(j) == text.charAt(i)) {
20                 i++;
21                 j++;
22             }
23
24             if (j == M) {
25                 System.out.println("Pattern found at index " + (i - j));
26                 j = lps[j - 1];
27             } else if (i < N && pattern.charAt(j) != text.charAt(i)) {
28                 if (j != 0) {
29                     j = lps[j - 1];
30                 } else {
31                     i++;
32                 }
33             }
34         }
35     }
36 }
```

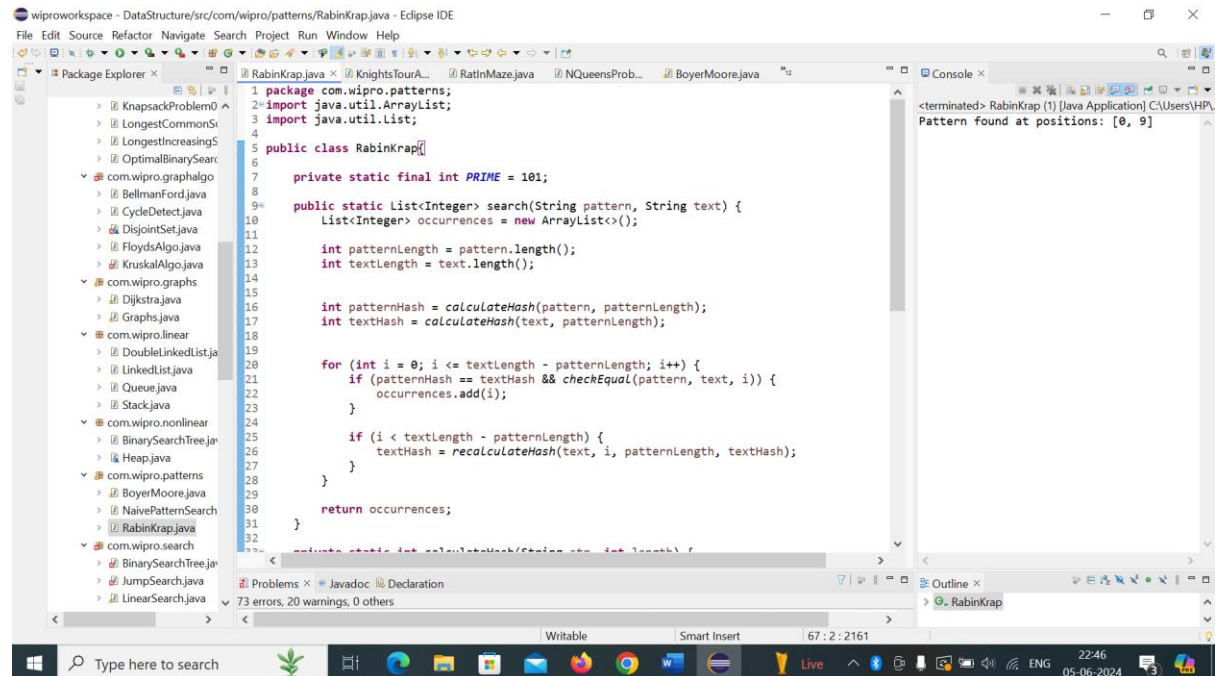


The screenshot shows the Eclipse IDE with the KMPAlgo.java file open, displaying the `computeLPSArray` method. This method calculates the Longest Prefix Suffix (LPS) array for a given pattern, which is used by the `search` method to efficiently find the pattern in the text.

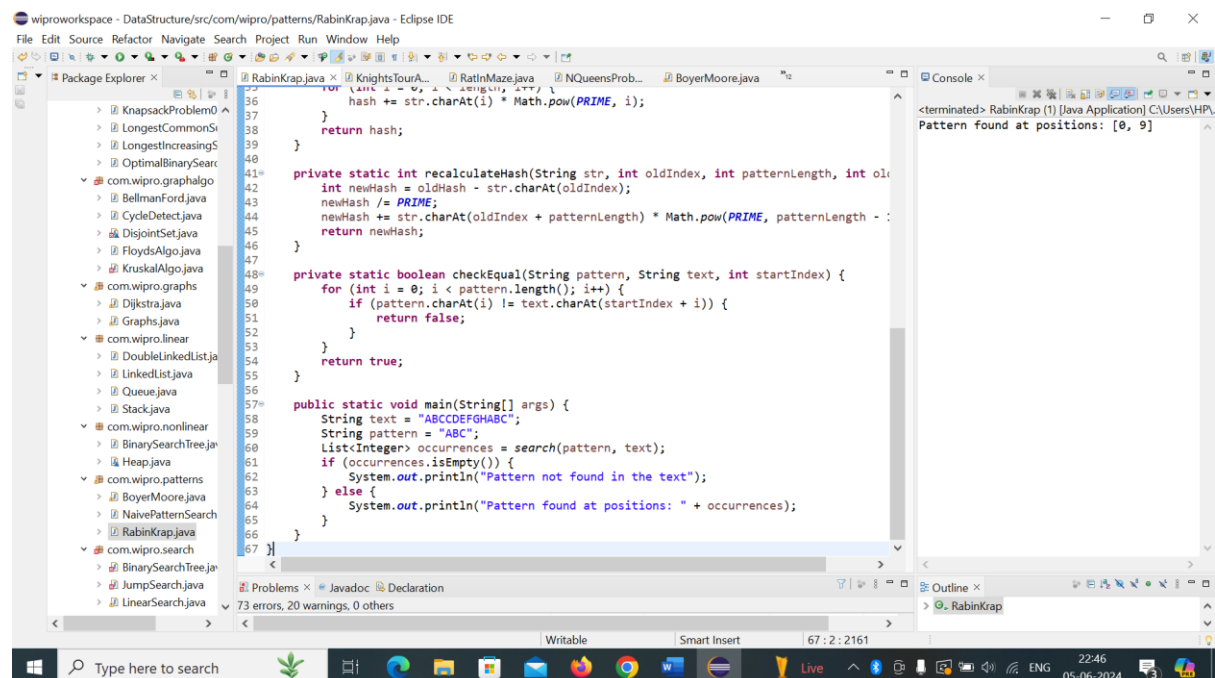
```
37 private static int[] computeLPSArray(String pattern) {
38     int M = pattern.length();
39     int[] lps = new int[M];
40     int length = 0;
41     int i = 1;
42     lps[0] = 0;
43
44     while (i < M) {
45         if (pattern.charAt(i) == pattern.charAt(length)) {
46             length++;
47             lps[i] = length;
48             i++;
49         } else {
50             if (length != 0) {
51                 length = lps[length - 1];
52             } else {
53                 lps[i] = 0;
54                 i++;
55             }
56         }
57     }
58     return lps;
59 }
60 }
```

## 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.



```
1 package com.wipro.patterns;
2 import java.util.ArrayList;
3 import java.util.List;
4
5 public class RabinKrap {
6
7     private static final int PRIME = 101;
8
9     public static List<Integer> search(String pattern, String text) {
10         List<Integer> occurrences = new ArrayList<>();
11
12         int patternLength = pattern.length();
13         int textLength = text.length();
14
15         int patternHash = calculateHash(pattern, patternLength);
16         int textHash = calculateHash(text, patternLength);
17
18         for (int i = 0; i < textLength - patternLength; i++) {
19             if (patternHash == textHash && checkEqual(pattern, text, i)) {
20                 occurrences.add(i);
21             }
22
23             if (i < textLength - patternLength) {
24                 textHash = recalculateHash(text, i, patternLength, textHash);
25             }
26         }
27
28         return occurrences;
29     }
30
31     private static int calculateHash(String str, int length) {
32         int hash = 0;
33         for (int i = 0; i < length; i++) {
34             hash += str.charAt(i) * Math.pow(PRIME, i);
35         }
36         return hash;
37     }
38
39     private static int recalculateHash(String str, int oldIndex, int patternLength, int oldHash) {
40         int newHash = oldHash - str.charAt(oldIndex);
41         newHash /= PRIME;
42         newHash += str.charAt(oldIndex + patternLength) * Math.pow(PRIME, patternLength - 1);
43         return newHash;
44     }
45
46     private static boolean checkEqual(String pattern, String text, int startIndex) {
47         for (int i = 0; i < pattern.length(); i++) {
48             if (pattern.charAt(i) != text.charAt(startIndex + i)) {
49                 return false;
50             }
51         }
52         return true;
53     }
54
55     public static void main(String[] args) {
56         String text = "ABCDEFHABC";
57         String pattern = "ABC";
58         List<Integer> occurrences = search(pattern, text);
59         if (occurrences.isEmpty()) {
60             System.out.println("Pattern not found in the text");
61         } else {
62             System.out.println("Pattern found at positions: " + occurrences);
63         }
64     }
65 }
66
67 }
```



```
36         int hash = 0;
37         for (int i = 0; i < length; i++) {
38             hash += str.charAt(i) * Math.pow(PRIME, i);
39         }
40         return hash;
41     }
42
43     private static int recalculateHash(String str, int oldIndex, int patternLength, int oldHash) {
44         int newHash = oldHash - str.charAt(oldIndex);
45         newHash /= PRIME;
46         newHash += str.charAt(oldIndex + patternLength) * Math.pow(PRIME, patternLength - 1);
47         return newHash;
48     }
49
50     private static boolean checkEqual(String pattern, String text, int startIndex) {
51         for (int i = 0; i < pattern.length(); i++) {
52             if (pattern.charAt(i) != text.charAt(startIndex + i)) {
53                 return false;
54             }
55         }
56         return true;
57     }
58
59     public static void main(String[] args) {
60         String text = "ABCDEFHABC";
61         String pattern = "ABC";
62         List<Integer> occurrences = search(pattern, text);
63         if (occurrences.isEmpty()) {
64             System.out.println("Pattern not found in the text");
65         } else {
66             System.out.println("Pattern found at positions: " + occurrences);
67         }
68     }
69 }
```

## 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.

wiproworkspace - DataStructure/src/com/wipro/patterns/BoyerMoore.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- > KnapsackProblem0
- > LongestCommonSi
- > LongestIncreasingS
- > OptimalBinarySear
- > com.wipro.graphalgo
  - > BellmanFord.java
  - > CycleDetect.java
  - > DisjointSet.java
  - > FloydAlgo.java
  - > KruskalAlgo.java
- > com.wipro.graphs
  - > Dijkstra.java
  - > Graphs.java
- > com.wipro.linear
  - > DoubleLinkedListJa
  - > LinkedList.java
  - > Queue.java
  - > Stack.java
- > com.wipro.nonlinear
  - > BinarySearchTreeJa
  - > Heap.java
- > com.wipro.patterns
  - > **BoyerMoore.java**
  - > NaivePatternSearch
  - > RabinKrap.java
- > com.wipro.search
  - > BinarySearchTreeJa
  - > JumpSearch.java
  - > LinearSearch.java

```
1 package com.wipro.patterns;
2 public class BoyerMoore{
3     public static int lastOccurrence(String txt, String pat) {
4         int[] badChar = badCharHeuristic(pat);
5         int[] goodSuffix = goodSuffixHeuristic(pat);
6
7         int m = pat.length();
8         int n = txt.length();
9
10        int s = 0;
11        int lastOccurrence = -1;
12
13        while (s <= n - m) {
14            int j = m - 1;
15
16            while (j >= 0 && pat.charAt(j) == txt.charAt(s + j)) {
17                j--;
18            }
19
20            if (j < 0) {
21                lastOccurrence = s;
22                s += (s + m < n) ? m - badChar[txt.charAt(s + m)] : 1;
23            } else {
24                s += Math.max(1, j - badChar[txt.charAt(s + j)]);
25            }
26        }
27
28        return lastOccurrence;
29    }
30
31    private static int[] badCharHeuristic(String pat) {
32        int[] badChar = new int[256];
33        for (int i = 0; i < pat.length(); i++) {
34            badChar[pat.charAt(i)] = i;
35        }
36        return badChar;
37    }
38}
```

Console

<terminated> BoyerMoore [Java Application] C:\Users\HP\AppData\Local\Temp\plugins\org.ecj...  
Last occurrence of pattern is at index 10

Outline

- BoyerMoore
- lastOccurrence(String, String) : int

Problems 73 errors, 20 warnings, 0 others

Writable Smart Insert 89 : 2 : 2387

wiproworkspace - DataStructure/src/com/wipro/patterns/BoyerMoore.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- > KnapsackProblem0
- > LongestCommonSi
- > LongestIncreasingS
- > OptimalBinarySear
- > com.wipro.graphalgo
  - > BellmanFord.java
  - > CycleDetect.java
  - > DisjointSet.java
  - > FloydAlgo.java
  - > KruskalAlgo.java
- > com.wipro.graphs
  - > Dijkstra.java
  - > Graphs.java
- > com.wipro.linear
  - > DoubleLinkedListJa
  - > LinkedList.java
  - > Queue.java
  - > Stack.java
- > com.wipro.nonlinear
  - > BinarySearchTreeJa
  - > Heap.java
- > com.wipro.patterns
  - > **BoyerMoore.java**
  - > NaivePatternSearch
  - > RabinKrap.java
- > com.wipro.search
  - > BinarySearchTreeJa
  - > JumpSearch.java
  - > LinearSearch.java

```
39 for (int i = 0; i < m; i++) {
40     badChar[pat.charAt(i)] = i;
41 }
42 return badChar;
43 }
44
45 private static int[] goodSuffixHeuristic(String pat) {
46     int m = pat.length();
47     int[] suffix = new int[m];
48     int[] shift = new int[m];
49
50     for (int i = 0; i < m; i++) {
51         suffix[i] = -1;
52     }
53
54     int lastPrefixPosition = m;
55     for (int i = m - 1; i >= 0; i--) {
56         if (isPrefix(pat, i + 1)) {
57             lastPrefixPosition = i + 1;
58         }
59         shift[m - 1 - i] = lastPrefixPosition - i + m - 1;
60     }
61
62     for (int i = 0; i < m - 1; i++) {
63         int slen = suffixLength(pat, i);
64         shift[slen] = m - 1 - i + slen;
65     }
66
67     return shift;
68 }
69
70 private static boolean isPrefix(String pat, int i) {
71     for (int j = 0; j < i; j++) {
72         if (pat.charAt(j) != pat.charAt(j + i)) {
73             return false;
74         }
75     }
76     return true;
77 }
78
79 private static int suffixLength(String pat, int i) {
80     int j = i;
81     while (j < pat.length() && pat.charAt(j) == pat.charAt(j + i)) {
82         j++;
83     }
84     return j - i;
85 }
86}
```

Console

<terminated> BoyerMoore [Java Application] C:\Users\HP\AppData\Local\Temp\plugins\org.ecj...  
Last occurrence of pattern is at index 10

Outline

- BoyerMoore
- lastOccurrence(String, String) : int

Problems 73 errors, 20 warnings, 0 others

Writable Smart Insert 89 : 2 : 2387

