DAY 17:

ASSIGNMENT 3:

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.

ANSWER:

```
import java.util.Arrays;


public class BoyerMoore {


    private static final int ALPHABET_SIZE = 256;


    // Function to preprocess the bad character heuristic array
    private static void preprocessBadCharacterHeuristic(char[] pattern, int m, int[] badChar) {
        Arrays.fill(badChar, -1);
        for (int i = 0; i < m; i++) {
            badChar[pattern[i]] = i;
        }
    }


    // Function to preprocess the good suffix heuristic arrays
    private static void preprocessGoodSuffixHeuristic(char[] pattern, int m, int[] suffix, int[] goodSuffix)
{
        Arrays.fill(suffix, -1);
        Arrays.fill(goodSuffix, m);


        int f = 0, g = m - 1;
        suffix[m - 1] = m;
        for (int i = m - 2; i >= 0; i--) {
            if (i > g && suffix[i + m - 1 - f] < i - g) {
```

```
        suffix[i] = suffix[i + m - 1 - f];
    } else {
        if (i < g) {
            g = i;
        }
        f = i;
        while (g >= 0 && pattern[g] == pattern[g + m - 1 - f]) {
            g--;
        }
        suffix[i] = f - g;
    }
}

for (int i = 0; i < m; i++) {
    goodSuffix[i] = m;
}

for (int i = 0, j = 0; j < m - 1; j++) {
    if (suffix[j] == j + 1) {
        while (i < m - 1 - j) {
            if (goodSuffix[i] == m) {
                goodSuffix[i] = m - 1 - j;
            }
            i++;
        }
    }
}

for (int i = 0; i < m - 1; i++) {
    goodSuffix[m - 1 - suffix[i]] = m - 1 - i;
}
```

```
    }

    // Function to find the last occurrence of a pattern in a text using Boyer-Moore algorithm
    public static int boyerMooreSearch(String text, String pattern) {
        char[] txt = text.toCharArray();
        char[] pat = pattern.toCharArray();
        int n = txt.length;
        int m = pat.length;

        int[] badChar = new int[ALPHABET_SIZE];
        int[] suffix = new int[m];
        int[] goodSuffix = new int[m];

        preprocessBadCharacterHeuristic(pat, m, badChar);
        preprocessGoodSuffixHeuristic(pat, m, suffix, goodSuffix);

        int s = 0; // s is the shift of the pattern with respect to text
        int lastOccurrence = -1;
        while (s <= (n - m)) {
            int j = m - 1;

            // Keep reducing j while characters of pattern and text are matching
            while (j >= 0 && pat[j] == txt[s + j]) {
                j--;
            }

            // If the pattern is present at the current shift
            if (j < 0) {
                lastOccurrence = s;
                s += goodSuffix[0]; // Shift the pattern so that the next character in text aligns with the last
occurrence of it in the pattern
```

```
            } else {
                s += Math.max(goodSuffix[j], j - badChar[txt[s + j]]);
            }
        }

        return lastOccurrence;
    }

    // Main function to test the algorithm
    public static void main(String[] args) {
        String text = "HERE IS A SIMPLE EXAMPLE HERE";
        String pattern = "HERE";
        int lastIndex = boyerMooreSearch(text, pattern);
        if (lastIndex != -1) {
            System.out.println("The last occurrence of pattern is at index: " + lastIndex);
        } else {
            System.out.println("Pattern not found");
        }
    }
}
```

EXPLANATION:

1. Bad Character Heuristic:

- The preprocessBadCharacterHeuristic function initializes the bad character array. This array records the last occurrence of each character in the pattern, helping to determine the shift when a mismatch occurs.

2. Good Suffix Heuristic:

- The preprocessGoodSuffixHeuristic function constructs the suffix and good suffix arrays. These arrays are used to determine the shift when a mismatch occurs at a suffix of the pattern.

3. Boyer-Moore Search:

- The boyerMooreSearch function searches for the pattern in the text using both the bad character and good suffix heuristics. It tracks the last occurrence of the pattern by updating the shift based on the values from these heuristics.

4. Main Function - This tests the boyerMooreSearch function with a sample text "HERE IS A SIMPLE EXAMPLE HERE" and the pattern "HERE". It prints the index of the last occurrence of the pattern in the text.

Efficiency of Boyer-Moore Algorithm

The Boyer-Moore algorithm can outperform other string searching algorithms due to its ability to skip large portions of the text. It is particularly effective when the pattern and text contain few common characters or when the pattern is long relative to the alphabet size. By leveraging both the bad character and good suffix heuristics, it reduces the number of character comparisons needed, resulting in faster search times in practical scenarios.