Day 59/180 String Matching KMP Algorithm

1: Find the Index of the First Occurrence in a String:

Logic and code both are explained in the video (Day 59).

2: Search Pattern (KMP-Algorithm):

Step 1 : Compute the Longest Prefix Suffix (LPS) Array:

- The function computeLPS calculates the Longest Prefix Suffix array for the given pattern pat. The LPS array is used to avoid unnecessary comparisons in the main pattern matching algorithm.
- It uses two pointers, i and len, to iterate through the pattern. The variable len keeps track of the length of the previous longest prefix suffix.
- If the characters at pat[i] and pat[len] match, it means a longer prefix suffix is found. Increment len and set lps[i] to len + 1.
- If there is a mismatch and len is not zero, set len to the previous value in the LPS array (lps[len-1]). Otherwise, set lps[i] to 0 and move to the next character in the pattern.

Step 2 : Pattern Search using KMP Algorithm:

- The search function performs the actual pattern search in the text txt using the pattern pat.
- It initializes variables i and j to 0 for traversing the text and the pattern, respectively.
- Inside the while loop, it checks if the characters at positions txt[i] and pat[j] match. If they do, both pointers are incremented.
- If there is a mismatch and j is not zero, it means there was a partial match. In this case, update j to the previous value in the LPS array (lps[j-1]).
- If there is a mismatch and j is already at the beginning of the pattern, increment i to continue searching in the text.

- If j reaches the end of the pattern, a complete match is found. Add the starting index of the match to the result vector ans, and update j to the previous value in the LPS array (lps[j-1]).
- Continue this process until the entire text is traversed.

Step 3 :Return the Result:

- If there are no matches (ans.size() is 0), return {-1}.
- Otherwise, return the vector ans, which contains the starting indices of all occurrences of the pattern in the text.

```
// Function to compute the Longest Prefix Suffix (LPS) array for a pattern
void computeLPS(string pat, vector<int> &lps) {
    int m = pat.size();

    // If the pattern has only one character, LPS is 0 (trivial case)
    if(m == 1) return;

    int len = 0, i = 1;

    // Loop to fill the LPS array
    while(i < m) {
        // If characters at positions i and len match
        if(pat[i] == pat[len]) {
            lps[i] = len + 1;
            len++;
            i++;
        } else {
            // If there is a mismatch
            if(len != 0) {
                // Move len to the previous value in the LPS array
                len = lps[len-1];
            } else {
                // Set LPS for the current position to 0 and move to the
next character
                lps[i] = 0;
                i++;
            }
        }
    }
}

// Function to search for occurrences of a pattern in a text using KMP
```

```
algorithm
vector<int> search(string pat, string txt) {
    int m = pat.size(), n = txt.size();

    // Initialize LPS array and result vector
    vector<int> lps(m, 0);
    vector<int> ans;

    // Compute LPS array for the pattern
    computeLPS(pat, lps);

    int i = 0, j = 0;

    // Main pattern matching loop
    while(i < n) {
        // If characters at positions j in pattern and i in text match
        if(pat[j] == txt[i]) {
            i++;
            j++;
        } else {
            // If there is a mismatch
            if(j != 0) {
                // Move j to the previous value in the LPS array
                j = lps[j-1];
            } else {
                // Move to the next character in the text
                i++;
            }
        }

        // If the entire pattern is matched, add the starting index to the
result vector
        if(j == m) {
            ans.push_back(i - j + 1);

            // Move j to the previous value in the LPS array to continue
searching
            j = lps[j-1];
        }
    }

    // If no matches found, return -1
    if(ans.size() == 0) return {-1};
```

```
    // Return the vector containing starting indices of pattern occurrences
    return ans;
}
```