KAZAKH KBTU BRITISH TECHNICAL UNIVERSITY

LECTURE 9 – KNUTH-MORRIS-PRATT FOR STRING MATCHING PROBLEMS

Prepared by: Izbassar Assylzhan

Course: Algorithms & Data Structures – Fall 2025

LECTURE 9

KAZAKH KBTU BRITISH TECHNICAL UNIVERSITY

In this lecture, we'll cover one of efficient string-matching algorithm Knuth-Morris-Pratt (KMP) for finding all the occurrences of some pattern with a text in O(n + m).

TOPICS WE'LL COVER:

- Comparison with naïve approach
- Longest Prefix Suffix
- String matching
- Practice Problems

GOALS FOR THIS LECTURE:

- Understanding the efficiency of the algorithm comparing with naïve approach;
- Implementing the calculation of longest prefix-suffix of the pattern;
- Implementing KMP algorithm & apply them to string matching problems.

COMPARISON WITH NAÏVE APPROACH

- Comparison with naïve approach
- Longest Prefix Suffix
- String matching
- Practice Problems

The *naive string-matching* algorithm checks the pattern at every text position, comparing characters sequentially.

On a mismatch, it shifts by one and restarts, causing rechecking the same characters repeatedly.

Example:

Searching for "aaaab" in "aaaaaaaab" results in many redundant comparisons, making the algorithm's time complexity O(n × m). [2]

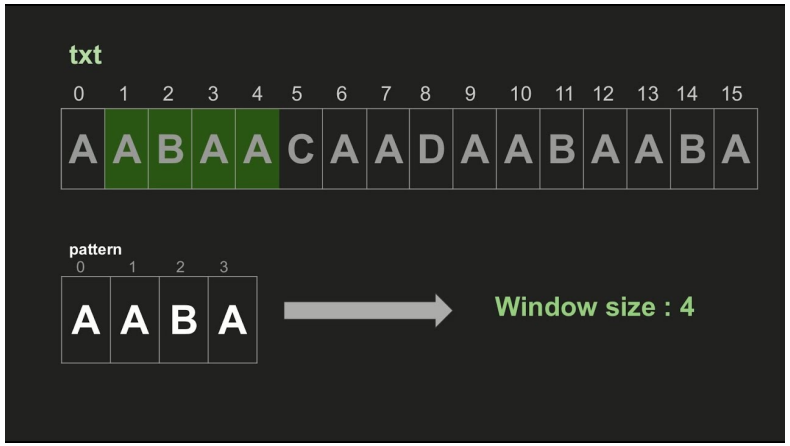

Figure 1 – Naïve string matching. [1]

The **KMP algorithm** *eliminates this inefficiency by preprocessing the pattern with a helper array called LPS (Longest Prefix Suffix).*

[1] https://www.youtube.com/watch?v=nK7SLhXcqRo;
[2] https://www.geeksforgeeks.org/dsa/kmp-algorithm-for-pattern-searching/

LONGEST PREFIX SUFFIX

KAZAKH KBTU BRITISH TECHNICAL UNIVERSITY

- Comparison with naïve approach
- Longest Prefix Suffix
- String matching
- Practice Problems

The LPS array records, for each position in the pattern, the length of the longest *proper prefix* that is also a suffix of the substring that ends at that position. [2]

✓ **Proper prefix** *is a prefix of pattern that is not equal to the pattern itself. For example, the proper prefix of "kbtu": "", "k", "kb", "kbt".*

Example: Pattern "ababaa"

- At index 0: "a" → No proper prefix/suffix → lps[0] = 0;
- At index 1: "ab" → No prefix matches suffix → lps[1] = 0;
- At index 2: "aba" → "a" is prefix & suffix → lps[2] = 1;
- At index 3: "abab" → "ab" is prefix & suffix → lps[3] = 2;
- At index 4: "ababa" → "aba" is prefix & suffix → lps[4] = 3;
- At index 5: "ababaa" → "a" is prefix & suffix → lps[5] = 1;

Finally: lps = {0, 0, 1, 2, 3, 1}

[2] https://www.geeksforgeeks.org/dsa/kmp-algorithm-for-pattern-searching/

LONGEST PREFIX SUFFIX (CONT.)

- Comparison with naïve approach
- Longest Prefix Suffix
- String matching
- Practice Problems

For constructing lps array, as 0 position, we set the value 0, because the string with length = 1, has no proper prefix. [2]

Then, we have to initialize a variable *len*, which stores the length of longest prefix that is also a suffix for the previous index. As we increment the index from 1, we compare the current char pat[i] with pat[len]. According to that comparison, we have 3 options:

Case 1: pat[i] == pat[len]

This indicates that the current character extends the ongoing prefix-suffix match.
- ✓ Increase len by 1 and set lps[i] = len.
- ✓ Proceed to the next index.

Case 2: pat[i] != pat[len] & len == 0

No prefix matches the suffix ending at position *i*, and there's no earlier match to reuse.
- ✓ Therefore, we set lps[i] = 0 and move on to the next character.

[2] https://www.geeksforgeeks.org/dsa/kmp-algorithm-for-pattern-searching/

LONGEST PREFIX SUFFIX (CONT.)

- Comparison with naïve approach
- Longest Prefix Suffix
- String matching
- Practice Problems

Case 3: pat[i] != pat[len] & len > 0

We can't extend the current prefix-suffix match, but a shorter matching prefix might still exist. Instead of checking all prefixes again, we use the previously calculated LPS values.

- ✓ Because pat[0...len-1] matches pat[i-len ... i-1], we update len to lps[len - 1].
- ✓ This effectively shortens the prefix we compare and **avoids repeating work**. We don't advance i yet — we recheck pat[i] using the updated len.

```cpp
vector<int> computeLPSArray(string& pattern) {
    int n = pattern.size();
    vector<int> lps(n, 0);

    int len = 0; // length of previous longest prefix = suffix
    int i = 1;

    while (i < n) {
        if (pattern[i] == pattern[len]) {
            len++;
            lps[i] = len;
            i++;
        } else {
            if (len != 0) {
                len = lps[len - 1]; // fall back in the pattern
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
    return lps;
}
```

Figure 2 – Finding the values of longest prefix-suffix.

[2] https://www.geeksforgeeks.org/dsa/kmp-algorithm-for-pattern-searching/

STRING MATCHING

KAZAKH KBTU BRITISH TECHNICAL UNIVERSITY

- Comparison with naïve approach
- Longest Prefix Suffix
- String matching
- Practice Problems

The KMP algorithm works in two main steps:

1. **Preprocessing Step** – Calculating LPS for pattern;
2. **Matching Step** – Searching the pattern in the text.

We compare the pattern and text character by character.

- **If they match:** move both pointers forward.
- **If they don't match:**
  → If not at the pattern's start, use lps[j - 1] to shift the pattern to the longest prefix that's also a suffix — skipping rechecks.
  → If at the start (j == 0), just move the text pointer ahead.
- **If the pattern ends:** a full match is found — record its starting position and keep searching.

```cpp
vector<int> search(string& pat, string& text) {
    vector<int> lps = computeLPSArray(pat), result;

    int m = pat.size();
    int n = text.size();
    int i = 0, j = 0;

    while (i < n) {
        if (text[i] == pat[j]) {
            i++;
            j++;

            if (j == m) {
                result.push_back(i - j);
                j = lps[j-1];
            }
        } else {
            if (j != 0) {
                j = lps[j-1];
            } else {
                i++;
            }
        }
    }
    return result;
}
```

Figure 3 – Matching pattern with text.

[2] https://www.geeksforgeeks.org/dsa/kmp-algorithm-for-pattern-searching/

KAZAKH KBTU BRITISH TECHNICAL UNIVERSITY

PRACTICE PROBLEMS

- Comparison with naïve approach
- Longest Prefix Suffix
- String matching
- Practice Problems

We have two tasks to solve:

- 1392. Longest Happy Prefix;
- 2185. Counting Words With a Given Prefix.

Q & A