

KMP / LPS Array

Failure function — never re-examine matched chars ■

● HARD

■ Hook

Naive pattern matching: $O(n \times m)$. Knuth-Morris-Pratt: $O(n+m)$. The key: when a mismatch occurs, **don't throw away what you already matched**. The LPS (Longest Proper Prefix which is also Suffix) array tells you exactly how far back to jump — reusing partial matches.

■ Core Intuition — The LPS Array

$\text{lps}[i]$ = length of longest proper prefix of $\text{pattern}[0..i]$ that is also a suffix. When mismatch at $\text{pattern}[j]$, don't reset j to 0. Instead: $j = \text{lps}[j-1]$. The text pointer i never moves backward.

■ Build LPS + KMP

```
pattern = 'AAACAAAA'
idx:      0123456 7
lps:      0120123 3
Build lps:
lps[0]=0 (always)
i=1,j=0: p[1]='A'==p[0]='A' → lps[1]=1, j=1
i=2,j=1: p[2]='A'==p[1]='A' → lps[2]=2, j=2
i=3,j=2: p[3]='C'!=p[2]='A' → j=lps[1]=1
          p[3]='C'!=p[1]='A' → j=lps[0]=0
          p[3]='C'!=p[0]='A' → lps[3]=0
...continue...
KMP search: text pointer never goes backward.
Mismatch at pattern[j]? j = lps[j-1]. (j=0? advance text i)
```

■ Code

```

// Build LPS array
int[] buildLPS(String pat) {
    int n=pat.length();
    int[] lps = new int[n];
    int len=0, i=1;
    while (i < n) {
        if (pat.charAt(i) == pat.charAt(len)) {
            lps[i++] = ++len;
        } else {
            if (len != 0) len = lps[len-1]; // jump back
            else lps[i++] = 0;
        }
    }
    return lps;
}

// KMP search
List<Integer> kmpSearch(String text, String pat) {
    int[] lps = buildLPS(pat);
    List<Integer> res = new ArrayList<>();
    int i=0, j=0;
    while (i < text.length()) {
        if (text.charAt(i) == pat.charAt(j)) { i++; j++; }
        if (j == pat.length()) {
            res.add(i-j); // found at index i-j
            j = lps[j-1];
        } else if (i<text.length() && text.charAt(i)!=pat.charAt(j)) {
            if (j!=0) j=lps[j-1]; else i++;
        }
    }
    return res;
}

```

■ TIME

$O(n + m)$

■ SPACE

$O(m)$

- ✓ lps[i] = longest proper prefix of pat[0..i] that is also a suffix
- ✓ Build lps in $O(m)$ — when mismatch, jump to lps[len-1]
- ✓ KMP: text pointer i never moves back
- ✓ Mismatch at pattern[j]: j = lps[j-1] (j=0: advance i)
- ✓ Time: $O(n+m)$ | Space: $O(m)$

■ MENTAL MODEL — Remember This Forever

KMP's insight: when you fail a match at position j in the pattern, you haven't wasted all that work. $lps[j-1]$ tells you: 'the first $lps[j-1]$ characters of the pattern already match the text here.' You restart the pattern comparison from $lps[j-1]$, not from 0. The text pointer never goes backward — that's the $O(n)$ guarantee.