# KMP

The KMP stands for Knuth–Morris–Pratt algorithm, which is a string search algorithm. The original purpose for KMP is to use roughly O(N) time complexity to search a substring in a very long string.

There are two steps in KMP implementation, the first part is to build the kmp\_table on the substring to be looked for. The second part is to use the kmp\_table in the search.

There is several ways to implement KMP algorithm. Here I choose the one in Geeks for geeks. This is to calculate the longest prefix which is also the suffix of the substring at any position in the target string.

<https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/>

In this paper, the KMP table is built in this way, we start the position of index 1, which with pointer to index 0 which is the head of the string, we compare the characters, if they match, we advance the pointer, and record in the current index (starting as 1), if not match we step back to the pointer p - 1, if p < 0, we set the current index as 0.

The following are some examples of KMP table

For the pattern “AAAA”,

lps[] is [0, 1, 2, 3]

For the pattern “ABCDE”,

lps[] is [0, 0, 0, 0, 0]

For the pattern “AABAACAABAA”,

lps[] is [0, 1, 0, 1, 2, 0, 1, 2, 3, 4, 5]

For the pattern “AAACAAAAAC”,

lps[] is [0, 1, 2, 0, 1, 2, 3, 3, 3, 4]

For the pattern “AAABAAA”,

lps[] is [0, 1, 2, 0, 1, 2, 3]

When we use the kmp table to compare the long string, if we advance to the next position in both source and target string, if not match, we reset the position in the target string to the kmp table in the previous position, if target position is 0 still not match, we move position in source string to next position.

## 28. Implement strStr()

Easy

Implement [strStr()](http://www.cplusplus.com/reference/cstring/strstr/" \t "_blank).

Return the index of the first occurrence of needle in haystack, or **-1** if needle is not part of haystack.

**Example 1:**

**Input:** haystack = "hello", needle = "ll"

**Output:** 2

**Example 2:**

**Input:** haystack = "aaaaa", needle = "bba"

**Output:** -1

**Clarification:**

What should we return when needle is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when needle is an empty string. This is consistent to C's [strstr()](http://www.cplusplus.com/reference/cstring/strstr/" \t "_blank) and Java's [indexOf()](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html" \l "indexOf(java.lang.String)" \t "_blank).

/// <summary>

/// Leet code #28. Implement strStr()

///

/// Easy

///

/// Implement strStr().

///

/// Return the index of the first occurrence of needle in haystack, or -1

/// if needle is not part of haystack.

///

/// Example 1:

///

/// Input: haystack = "hello", needle = "ll"

/// Output: 2

///

/// Example 2:

///

/// Input: haystack = "aaaaa", needle = "bba"

/// Output: -1

/// Clarification:

///

/// What should we return when needle is an empty string? This is a great

/// question to ask during an interview.

///

/// For the purpose of this problem, we will return 0 when needle is an empty

/// string. This is consistent to C's strstr() and Java's indexOf().

/// </summary>

int LeetCodeString::strStr(string haystack, string needle)

{

if (haystack.size() < needle.size()) return -1;

// Step 1: build kmp table

vector<int> kmp\_table(needle.size());

int i = 1;

int j = 0;

while (i < (int)needle.size())

{

if (needle[i] == needle[j])

{

j++;

kmp\_table[i] = j;

i++;

}

else if (j == 0)

{

kmp\_table[i] = 0;

i++;

}

else

{

j = kmp\_table[j - 1];

}

}

// Step 2: search substring

i = 0;

j = 0;

while (i < (int)haystack.size())

{

if (j == needle.size())

{

break;

}

else if (haystack[i] == needle[j])

{

i++;

j++;

}

else if (j == 0)

{

i++;

}

else

{

j = kmp\_table[j - 1];

}

}

if (j == needle.size())

{

return i - needle.size();

}

else

{

return -1;

}

}