(../index.html)

Data Structures

Previous (comparison-of-search-trees.html) Next (tries.html)

Knuth-Morris-Pratt Algorithm

KMP Algorithm is one of the most popular patterns matching algorithms. KMP stands for Knuth Morris Pratt. KMP algorithm was invented by **Donald Knuth** and **Vaughan Pratt** together and independently by **James H Morris** in the year 1970. In the year 1977, all the three jointly published KMP Algorithm. KMP algorithm was the first **linear time complexity** algorithm for string matching.

KMP algorithm is one of the string matching algorithms used to find a Pattern in a Text.

KMP algorithm is used to find a **"Pattern"** in a **"Text"**. This algorithm campares character by character from left to right. But whenever a mismatch occurs, it uses a preprocessed table called **"Prefix Table"** to skip characters comparison while matching. Some times prefix table is also known as **LPS Table**. Here LPS stands for **"Longest proper Prefix which is also Suffix"**.

Steps for Creating LPS Table (Prefix Table)

Step 1 - Define a one dimensional array with the size equal to the length of the Pattern. (LPS[size])

Step 2 - Define variables **i & j**. Set i = 0, j = 1 and LPS[0] = 0.

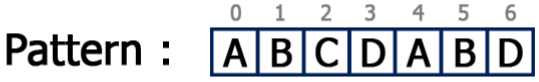
Step 5 - If both are not matched then check the value of variable 'i'. If it is '0' then set **LPS[j] = 0** and increment 'j' value by one, if it is not '0' then set **i = LPS[i-1]**. Goto Step 3.

Step 6- Repeat above steps until all the values of LPS[] are filled.

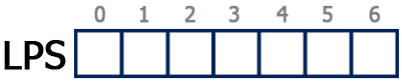
Let us use above steps to create prefix table for a pattern...

Example for creating KMP Algorithm's LPS Table (Prefix Table)

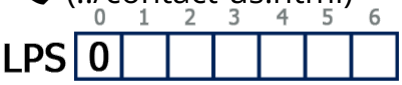
Consider the following Pattern



Let us define LPS[] table with size 7 which is equal to length of the Pattern

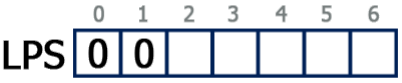


Step 1 - Define variables i & j. Set i = 0, j = 1 and LPS[0] = 0.



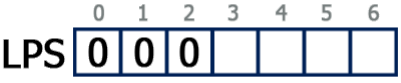
i = 0 and j = 1

Step 2 - Campare Pattern[i] with Pattern[j] ===> A with B.
Since both are not matching and also "i = 0", we need to set LPS[j] = 0 and increment 'j' value by one.



i = 0 and j = 2

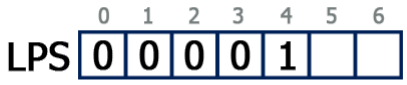
Step 3 - Campare Pattern[i] with Pattern[j] ===> A with C.
Since both are not matching and also "i = 0", we need to set LPS[j] = 0 and increment 'j' value by one.



i = 0 and j = 3

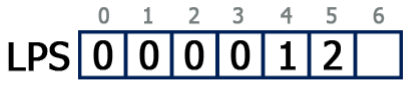
Step 4 - Campare Pattern[i] with Pattern[j] ===> A with D.
Since both are not matching and also "i = 0", we need to set LPS[j] = 0 and increment 'j' value by one.

Step 5 - Campare Pattern[i] with Pattern[j] ===> A with A.
Since both are matching set LPS[j] = i+1 and increment both i & j value by one.



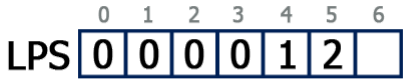
i = 1 and j = 5

Step 6 - Campare Pattern[i] with Pattern[j] ===> B with B.
Since both are matching set LPS[j] = i+1 and increment both i & j value by one.



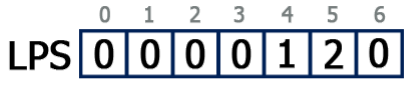
i = 2 and j = 6

Step 7 - Campare Pattern[i] with Pattern[j] ===> C with D.
Since both are not matching and i !=0, we need to set i = LPS[i-1]
===> i = LPS[2-1] = LPS[1] = 0.

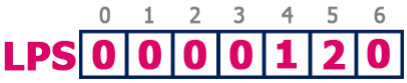


i = 0 and j = 6

Step 8 - Campare Pattern[i] with Pattern[j] ===> A with D.
Since both are not matching and also "i = 0", we need to set LPS[j] = 0 and increment 'j' value by one.



Here LPS[] is filled with all values so we stop the process. The final LPS[] table is as follows...



How to use LPS Table

We use the LPS table to decide how many characters are to be skipped for comparison when a mismatch has occurred.

When a mismatch occurs, check the LPS value of the previous character of the mismatched character in the pattern. If it is '0' then start comparing the first character of the pattern with the next character

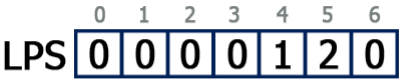
How the KMP Algorithm Works

Let us see a working example of KMP Algorithm to find a Pattern in a Text...

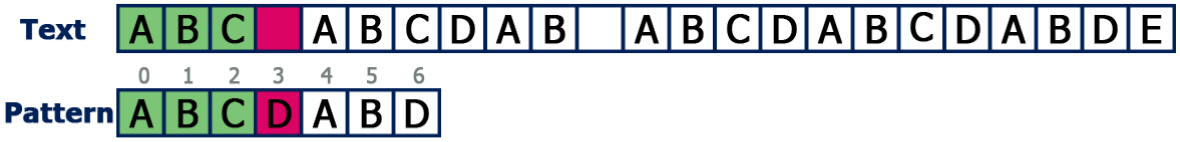
Consider the following Text and Pattern

Text : ABC ABCDAB ABCDABCDABDE
Pattern : ABCDABD

LPS[] table for the above pattern is as follows...

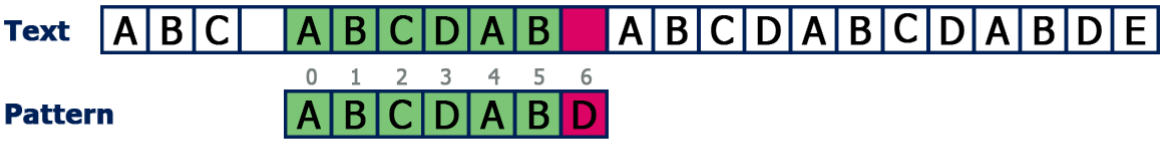


Step 1 - Start comparing first character of Pattern with first character of Text from left to right



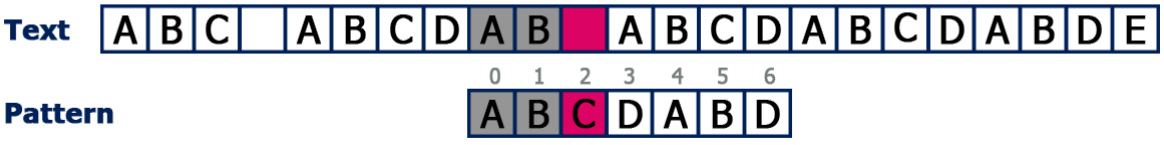
Here mismatch occured at Pattern[3], so we need to consider LPS[2] value. Since LPS[2] value is '0' we must compare first character in Pattern with next character in Text.

Step 2 - Start comparing first character of Pattern with next character of Text.

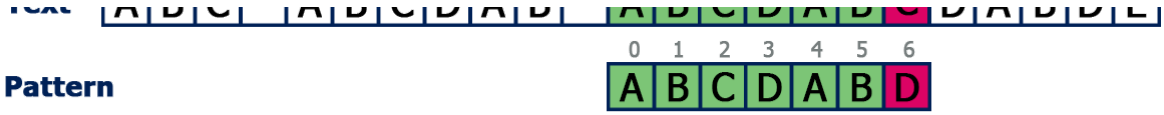


Here mismatch occured at Pattern[6], so we need to consider LPS[5] value. Since LPS[5] value is '2' we compare Pattern[2] character with mismatched character in Text.

Step 3 - Since LPS value is '2' no need to compare Pattern[0] & Pattern[1] values

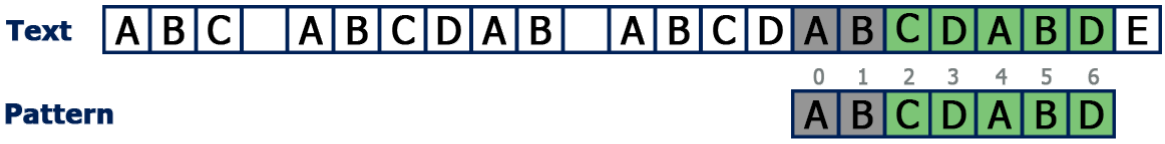


Here mismatch occured at Pattern[2], so we need to consider LPS[1] value. Since LPS[1] value is '0' we must compare first character in Pattern with next character in Text



Here mismatch occured at Pattern[6], so we need to consider LPS[5] value. Since LPS[5] value is '2' we compare Pattern[2] character with mismatched character in Text.

Step 5 - Compare Pattern[2] with mismatched character in Text.



Here all the characters of Pattern matched with a substring in Text which is starting from index value 15. So we conclude that given Pattern found at index 15 in Text.

Previous (comparison-of-search-trees.html) Next (tries.html)