

KMP Algo.

The purpose of this algorithm is to efficiently match patterns in string.

WORST CASE PERFORMANCE: $O(m+n)$

Suppose we have a string "ab^sdabcab^tdgabcabg" and a pattern "abcab^tg".
What KMP does is:

S: ab^sdabcab^tdgabcabg

t: abcab^tg

we try to match the two strings from the start.

There's a difference when comparing $S[2]$ and $t[2]$

We have to start over and compare $S[2]$, $t[0]$

Still, there is no match. Thus, we need to go one step further and get.

S: ab^sdabcab^tdgabcabg

t: abcab^tg

This time, we get something different!

The part that is matched, i.e. **abcab**, has the same prefix and suffix of length 2.

S: ab^sdabcab^tdgabcabg

t: abcab^tg

Knowing this, we can start matching from $t[2]$, as we know the last two letters is our prefix!

THIS IS THE GIST OF KMP ALGO!

To implement this idea, we first preprocess the string to construct a table (list)

pattern: a b c d g a b c f a b c a

p l: 0

The first one is always 0.

p: a b c d g a b c f a b c a

l: 0 0

↓ p: a b c d g a b c f a b c a

l: 0 0 0 0 0 1

↓

i, j starts as 0

now i goes to 1

if $p[i] \neq p[j]$
put 0 at $l[i]$
increment i.

now $p[i] = p[j]$.

put $j+1$ at $l[i]$,
increment i and j.

$j \rightarrow j \rightarrow j$ $i \rightarrow i \rightarrow i$
 a b c d g a b c f a b c a
 0 0 0 0 0 1 2 3

ANOTHER MISMATCH!

j will be reset to $d[j-1]$, which is 0.

$j \rightarrow j \rightarrow j \rightarrow j$ $i \rightarrow i \rightarrow i \rightarrow i$
 a b c d g a b c f a b c a
 0 0 0 0 0 1 2 3 0 1 2 3 1

TIME COMPLEXITY : $O(N)$

SPACE : $O(N)$

CAVEATS :

Consider : a a b a a b a a a
 0 1 0 1 2 3 4 5 $1+1=2$

Now $p[j] \neq p[i]$, so j needs to be set back to $d[j-1]=2$.

However, $p[2] \neq p[i]$, in this case we need to further set j to be $d[j-1]$

Use the table we built:

s: a b x a b c a b c a b y
 p: a b c a b y
 d: 0 0 0 1 2 0

The value at c is 0, which means in the next round we compare s with $p[0]$

s: a b x a b c a b c a b y
 p: a b c a b y
 d: 0 0 0 1 2 0

s: a b x a b c a b c a b y
 p: a b c a b y
 d: 0 0 0 1 2 0

Value is 2, next comparison would be $p[2]$.

s: a b x a b c a b c a b y
 p: a b c a b y
 d: 0 0 0 1 2 0

Match!