Text: \( T = T[1] \ T[2] \ldots \ T[n] \)

Pattern: \( P = P[1] \ P[2] \ldots \ P[m] \)

Find all occurrences of \( P \) in \( T \).

Idea:
Naive Algorithm

For i = 1 to n-m+1 do
    match ← 1
    For j = 1 to m do
        If \( P[j] \neq T[i+j-1] \) then
            match ← 0
    End
    If match = 1 then output "match at location" i
End

Time: \( O(nm) \)
EXAMPLE:

T:  
ABCD

P:  
ABCDE

Does it make sense to try:

T:  
ABCD

P:  
ABCDE

T:  
ABCD

P:  
ABCDE

T:  
ABCD

P:  
ABCDE

- Obviously Not!
Another Example:

T: ABCABC ABC A
P: ABCABC ABCD

Where is the next place to try?

T: ABCABC ABC A
P: ABCABC ABCD

And the next?

T: ABCABC ABC A
P: ABCABC ABCD

What is the rule?
The longest proper prefix of the pattern that is a suffix of the text.

**Problem:** This needs to be calculated for every text location many times. Can it be done fast enough?
ANSWER: Build a table and get this longest proper prefix in constant time.

Size of table: $O(m)$

Why? Transitivity of "=".

For every pattern location $i$ give largest proper pattern prefix that is also a pattern suffix.
When situation is:

\[ T: \]

\[ P: \]

We have from table the largest prefix that is also a pattern suffix.

But

\[ P[1] = T[i] \]
\[ P[2] = T[i+1] \]
\[ \vdots \]
\[ P[j-1] = T[i+j-2] \]

So by transitivity, the largest proper prefix of \( P \) that is a suffix of \( P \) ending at \( P[j-1] \) is also the largest prefix of \( P \) that is a suffix of \( T \) ending at \( T[i+j-2] \).
Construct automaton whose forward arrows are success links and whose back arrows are failure links.

The Success Links: (For automaton of pattern $P[i]... P[m]$)

```
0 1 2 ... m-1 m
```

The Failure Links:

The failure link from node $i$ points to node $j$, where $j$ is the length of the longest proper prefix of $P$ that is a suffix of $P$ ending at $P[i]$. 
Algorithm: Run on text with automaton. For success link, move to next text location. For failure, move on link but stay on text location.

EXAMPLE RUN:

Time:

Every move on success link was also a move forward in text. So how many times have we moved on success links?

$O(n)$

Since we never move back!

How many times do we move on failure links?
- Could be up to \( m \) in a row until start node is reached. Then forward again.

But in fail links we do not advance on text.

Does this mean that the time is \( O(nm) \)?

**KEY POINT:** For every failure link that we follow back, we had to go forward with a success link!

So total \# of fail links followed is not more than total \# of success link followed, i.e. \( O(n) \).
FORMALIZE

Define a counter $F$ that is initialized to 0.

Increment $F$ by 1: When a success link is followed.

Decrement $F$ by 1: When a failure link is followed.

Claim: At any point in the algorithm run, if the automaton is at node $k$ then $F \geq k$. 
Proof: By induction on the number of moves $i$ of the automaton algorithm.

Base case: $i = 0$

We are at node $0$, $F = 0$.

Induction Hypothesis: After move $i$, if the automaton is in node $k$ then $F \geq k$.

Prove for move $i+1$.

Cases: 1) Success link followed in move $i+1$.

\[
\begin{align*}
&k-1 \rightarrow k \\
&F \geq k-1 \\
&(\text{ind. hyp.}) \\
&F \leftarrow F+1 \\
&\Rightarrow F \geq k
\end{align*}
\]
2) Failure link followed in move $i+1$

\[ F \geq j \quad \text{(ind hyp)} \]
\[ j > k \]

\[ F \leftarrow F - 1 \quad \text{(fail link followed)} \]

\[ F \geq k \]

Conclude: $F$ is incremented at most $n$ times and is always non-negative. Therefore it is decremented (fail links) at most $n$ times.
Total Time Text Scan for KMP Algorithm

\( O(n) \).

How do we construct fail links?

Same method as search:

If \( a = b \) then failure link of \( k \) points to \( i+1 \).

Otherwise follow fail link from \( i \) and repeat...

Time: \( O(m) \).
Why?

Similar argument to text scan (can move back only if first moved forward).

Conclude: Total kmp time:

\( O(n+m) = O(n) \).