CS311 Yoshii - String Matching (No time to cover this during the summer)

String matching is a very frequently encountered problem
in Computer Science. For example, you might search through
a database or a web page looking for a string.

In AI, for **information retrieval**
and **building the language model in Natural Language Processing**, it is an essential task.

Text: **S = s1 s2 s3 ..... sn**

Pattern: **P = p1 p2 p3 ... pm**
where n >= m

Question: **Is P a substring of S? If so, where is its first occurrence?**

We will assume that the text is of length **n**
and the pattern is of length **m.**

Straight Forward Approach

The straightforward approach slides P over S (one character shift at a time)
as soon as a mismatch is found.

```
for i = 1 to n  // for each character position in S as the starting point of matching
{
  try to match p1...pm to  si...si+m-1
            (and stop matching as soon as mismatch is found to go to next i)
  If it is a match, return i;
}
```

Analysis:
Matching p1..pm against si..si+m-1, character by character, takes Theta(m)
in the worst case.
Since this is repeated n times (i = 1 to n) in the worst case, Theta(nm)

Inter1 What is this worst case requiring m character comparisons? Describe.

Inter2 Can this worst case occur every time through the for-loop?
Give an example to explain.
i.e. I = 1 suc suc suc fail I = 2 suc suc suc fail I = 3 suc suc suc fail.....

Notice that each time P is slid by one character,
the same M-1 characters in S must be compared again. Hm…….
Let's not slide by one character!!!

More Efficient String Algorithm: Knuth-Morris-Pratt Algorithm

Let's say

a part of the pattern
p1..pj (j characters) matched sk-j +1...sk (j characters)

e.g. p1..p3 matched s7..s9

s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ……
[p1 p2 p3 p4 ..]

But pj+1 does not match sk+1 (i.e. p4 does not match s10)

We want a failure function that brings us back to
the best possible state from which we can continue. i.e.
we don't want to slide P by 1 character.

e.g.

S = ... a b b a b b.....
P = [a b b a b a...]
p1 ...p5 matched **(j is 5** in this case)

Sliding P by one character does not help at all!
S = ...a b b a b b.....
P = [a b b a b a...]
Notice that a mismatch occurs right away.

So, let's slide P by 2 characters.
S = ...a b b a b b.....
P = [a b b a b a...]
Again, a mismatch occurs right away.

It is better to slide P by 3 characters so that p1 "a" is lined up
under "a" of S.
S = ...a b b a b b.....
P = [a b b a b a] Now abb matches abb

KMP: How do we determine how many characters to shift????

Recall that p1 through pj has already matched the characters of S.
But pj+1 did not match.

What we want is the longest head of p1..pj which matches
the tail of the matched part of S.

But p1..pj and sk-j+1...sk are the same!
==> So, what we want is the longest the head of p1...pj matching the tail of p....pj
Let's call the matched part p1 .. pj as P'.

---

e.g. P = a a b b a a b

matched P'	head matching tail	length
j=1 a	nothing matches but itself	0
j=2 aa	1st a matches second a	1
j=3 aab	no head matches b or ab	0
j=4 aabb	no head matches b,bb or abb	0
j=5 aabba	1st a matches last a	1
j=6 aabbaa	2a's match 2a's	2
j=7 aabbaab	aab matches aab	3

Therefore, here are the **fail function f** values:

f(j) means failure occurred with the j+1st char of P.

f(1) = 0 re-try 1st char of P against the mismatched character of S
f(2) = 1 re-try 2nd char of P
f(3) = 0 re-try 1st char of P
f(4) = 0 re-try 1st char of P
f(5) = 1 re-try 2nd char of P
f(6) = 2 re-try 3rd char of P
f(7) = 3 re-try 4th char of P

In General:
Given f(j) = h,
- p1...ph is the longest head of P' that matched the tail of P'
- The mismatch occurred at j+1st character of P with s"
=> We need to re-try the same s" character with the h+1st char of P

Examples of Difference Cases with the same P

P = a a b b a a b d (in all examples below, s' is 'c')

Eg1) matched p1=a mismatched p2=a
S a c …
P a a …
P' = a f(1) = 0 0+1 = 1
S a c …
P a a … **try p1 against 'c'**

Eg2) matched p1=a and p2=a mismatched p3=b
S a a c …
P a a b …
P' = aa f(2) = 1 1+1 = 2
S a a c …
P a a b … **try p2 against 'c'**

Eg3) matched p1=a p2=a p3=b mismatched p4=b
S a a b c …
P a a b b …
P' = aab f(3) = 0 0+1 = 1
S a a b c …
P a a b b … **try p1 against 'c'**

Eg4) matched p1=a p2=a p3=b p4=b mismatched p5=a
S a a b b c …
P a a b b a …
P' = aabb f(4) = 0 0+1 = 1
S a a b b c …
P a a b b … **try p1 against 'c'**

Eg5) matched p1=a p2=a p3=b p4=b p5=a mismatched p6=a
S a a b b a c …
P a a b b a a …
P' = aabba f(5) = 1 1+1 = 2
S a a b b a c …
P a a b b a a … **try p2 against 'c'**

KMP: Analysis of Pattern Matching

In the worst case,
- the number of successful comparisons is n (every character in S had to be compared)

- The number of unsuccessful comparisons is n (n failures)
This becomes clear if you build a finite state machine (CS421) and see where you go back to if a match fails. ***Finite State Machines are very important in CS.***

At most N+N = 2N comparisons total.
Compare this with Theta(NM) of the first approach.

As long as M > 2 (pattern length is > 2), KMP wins!!!