Knuth-Morris-Pratt Algorithm

→ previous pattern-matching algorithms we looked at had major inefficiencies when it came to the worst-case situation
  → for a certain alignment of the pattern, if we find several matching chars then a mismatch we ignore all info gained by successful comparisons by restarting
→ KMP algorithm avoids this waste of information and achieves $O(n+m)$ running time
  → worst case of any pattern matching using KMP will examine all characters of the text and of the pattern at least once
  → main idea of KMP is to precompute self-overlaps between portions of the pattern so that when a mismatch occurs at one location we know max amt. to shift pattern before continuing

→ Failure Function

→ precompute a failure function which will indicate the proper shift of P upon a failed comparison
→ specifically: f(k) defined as length of the longest prefix of P that is a suffix of P[1:k+1]
→ if we find a mismatch upon character P[k+1], the function tells us how many of the immediately preceding characters can be used to restart the pattern

```
def find_kmp(T, P):
  n, m = len(T), len(P)
  if m == 0: return 0
  fail = compute_kmp_fail(P)
  j, k = 0
  while j < n:
    if T[j] == P[k]:              // P[0:1+k] matched thus far
      if k == m-1:                // match complete
        return j-m+1
      j+=1, k+=1                  // try extend match
    elif k > 0:
      k = fail[k-1]               // reuse suffix of P[0:k]
    else:
      j+=1
  return -1                       // reached end w/o match
```