32. String Matching

There four string matching algorithms that will be discussed. Each of them differ in performance.

Algorithm	Preprocessing Time	Processing Time
Naive	0	$O((n-m+1)m)$
Rabin Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\|\sum\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Omega(m)$	$\Theta(n)$

Where, **n** is length of term, **m** is length of pattern, and **Σ** is number of alphabet that formed pattern.

Naive string-matching algorithm

The naive algorithm finds all possible matched for each character in pattern to each character in term. Because of this, total processing time is $O((n-m+1)m)$. The algorithm described below:

```
n = t.length
m = p.length
for i=0 to n:
  if [tᵢ, …, tᵢ₊ₘ] == [p₀,…, pₘ]
    // Matched
```

Exercise:

32.1-1 Show the comparisons the naive string matcher makes for the pattern P = 0001 in the text T = 00010001010001.

Matched in shift: 1, 5, 11

32.1-2 Suppose that all character in the pattern P are different. Show how to accelerate naive string matcher to run in time $O(n)$ on n-character text T.

```
n = t.length
m = p.length
k = m - 1
for i=0 to n:
  if j < m:
    if t[i] == p[j]:
      if j == k:
        // Matched
        j = 0
      j++
    else:
      if j > 0
        j = 0
```

32.1-3 Suppose that pattern P and text T are randomly choosen strings of length m and n, respectively, form *d*-array alphabet $\Sigma_d = \{0, 1, \ldots, d-1\}$, where $d >= 2$. Show that the expected number of character-to-character made by the implicit loop in line 4 of the naive algorithm is:

$$(n-m+1)\frac{1-d^{-m}}{1-d^{-1}} \leq 2(n-m+1)$$

over all execution of this loop. (Assume that the naive algorithm stops comparing characters for a given shift once it finds a mismatch or matches the entire pattern.) Thus, for randomly chosen strings, the naive algorithm is quite efficient.

32.1-4 Suppose we allow the pattern P to contain occurrences of **gap character** - that match an arbitrary string of characters (even one of zero length). For example, the pattern *ab-ba-c* occurs in the text *cabccbacbacab* as:

c ab cc ba cba c ab
  ab - ba - c

and as

c ab cccbac ba _____ c ab
  ab - ba - c

Note that the gap character may occur an arbitrary number of times in the pattern but not at all in the text. Give a polynomial-time algorithm to determine whether such a pattern P occurs in a given text T, and analyze the running time of your algorithm

Rabin-Karp algorithm