32 String Matching

32.1 The naive string-matching algorithm

32.1-1

> Show the comparisons the naive string matcher makes for the pattern $P = 0001$ in the text $T = 000010001010001$.

```
STRING-MATCHER(P, T, i)
    for j = i to i + P.length
        if P[j - i + 1] ≠ T[j]
            return false
    return true
```

32.1-2

> Suppose that all characters in the pattern P are different. Show how to accelerate $\text{NAIVE-STRING-MATCHER}$ to run in time $O(n)$ on an n-character text T.

Suppose $T[i] \ne P[j]$, then for $k \in [1, j)$, $T[i-k] = P[j-k] \ne P[0]$, the $[i-k, i)$ are all invalid shifts which could be skipped, therefore we can compare $T[i]$ with $P[0]$ in the next iteration.

32.1-3

> Suppose that pattern P and text T are randomly chosen strings of length m and n, respectively, from the d-ary alphabet $\Sigma_d = \{0, 1, \dots, d-1\}$, where $d \ge 2$. Show that the expected number of character-to-character comparisons made by the implicit loop in line 4 of the naive algorithm is
>
> $$(n-m+1)\frac{1-d^{-m}}{1-d^{-1}} \le 2(n-m+1)$$
>
> over all executions 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.

Suppose for each shift, the number of compared characters is L, then:

$$\begin{aligned} E[L] &= 1 \cdot \frac{d-1}{d} + 2 \cdot (\frac{1}{d})^1 \frac{d-1}{d} + \cdots + m \cdot (\frac{1}{d})^{m-1} \frac{d-1}{d} + m \cdot (\frac{1}{d})^m \\ &= (1 + 2 \cdot (\frac{1}{d})^1 + \cdots + m \cdot (\frac{1}{d})^m) \frac{d-1}{d} + m \cdot (\frac{1}{d})^m. \end{aligned}$$

$$\begin{aligned} S &= 1 + 2 \cdot (\frac{1}{d})^1 + \cdots + m \cdot (\frac{1}{d})^{m-1} \\ \frac{1}{d} S &= 1 \cdot (\frac{1}{d})^1 + \cdots + (m-1) \cdot (\frac{1}{d})^{m-1} + m \cdot (\frac{1}{d})^m \\ \frac{d-1}{d} S &= 1 + (\frac{1}{d})^1 + \cdots + \cdot (\frac{1}{d})^{m-1} - m \cdot (\frac{1}{d})^m \\ \frac{d-1}{d} S &= \frac{1-d^{-m}}{1-d^{-1}} - m \cdot (\frac{1}{d})^m. \end{aligned}$$

$$\begin{aligned} E[L] &= (1 + 2 \cdot (\frac{1}{d})^1 + \cdots + m \cdot (\frac{1}{d})^m)\frac{d-1}{d} + m \cdot (\frac{1}{d})^m \\ &= \frac{1 - d^{-m}}{1 - d^{-1}} - m \cdot (\frac{1}{d})^m + m \cdot (\frac{1}{d})^m \\ &= \frac{1 - d^{-m}}{1 - d^{-1}}. \end{aligned}$$

There are $n - m + 1$ shifts, therefore the expected number of comparisons is:

$$(n - m + 1) \cdot E[L] = (n - m + 1)\frac{1 - d^{-m}}{1 - d^{-1}}$$

Since $d \ge 2$, $1 - d^{-1} \ge 0.5$, $1 - d^{-m} < 1$, and $\frac{1-d^{-m}}{1-d^{-1}} \le 2$, therefore

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

32.1-4

> Suppose we allow the pattern P to contain occurrences of a ***gap character*** $\diamond$ that can match an *arbitrary* string of characters (even one of zero length). For example, the pattern $ab \diamond ba \diamond c$ occurs in the text $cabccbacbacab$ as
>
> $$c\underbrace{ab}_{ab}\underbrace{cc}_{\diamond}\underbrace{ba}_{ba}\underbrace{cba}_{\diamond}\underbrace{c}_{c}ab$$
>
> and as
>
> $$c\underbrace{ab}_{ab}\underbrace{ccbac}_{\diamond}\underbrace{ba}_{ba}\underbrace{\quad}_{\diamond}\underbrace{c}_{c}ab$$
>
> 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.

By using dynamic programming, the time complexity is $O(mn)$ where m is the length of the text T and n is the length of the pattern P; the space complexity is $O(mn)$, too.

This problem is similar to LeetCode 44. WildCard Matching, except that it has no question mark (?) requirement. You can see my naive DP implementation here.

32.2 The Rabin-Karp algorithm

32.2-1

> Working modulo $q = 11$, how many spurious hits does the Rabin-Karp matcher encounter in the text $T = 3141592653589793$ when looking for the pattern $P = 26$?

$|\{15, 59, 92\}| = 3$.

32.2-2

> How would you extend the Rabin-Karp method to the problem of searching a text string for an occurrence of any one of a given set of k patterns? Start by assuming that all k patterns have the same length. Then generalize your solution to allow the patterns to have different lengths.

Truncation.

32.2-3

> Show how to extend the Rabin-Karp method to handle the problem of looking for a given $m \times m$ pattern in an $n \times n$ array of characters. (The pattern may be shifted vertically and horizontally, but it may not be rotated.)

Calculate the hashes in each column just like the Rabin-Karp in one-dimension, then treat the hashes in each row as the characters and hashing again.

32.2-4

> Alice has a copy of a long n-bit file $A = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$, and Bob similarly has an n-bit file $B = \langle b_{n-1}, b_{n-2}, \ldots, b_0 \rangle$. Alice and Bob wish to know if their files are identical. To avoid transmitting all of A or B, they use the following fast probabilistic check. Together, they select a prime $q > 1000n$ and randomly select an integer x from $\{0, 1, \ldots, q - 1\}$. Then, Alice evaluates
>
> $$A(x) = (\sum_{i=0}^{n-1} a_i x^i) \mod q$$
>
> and Bob similarly evaluates $B(x)$. Prove that if $A \ne B$, there is at most one chance in 1000 that $A(x) = B(x)$, whereas if the two files are the same, $A(x)$ is necessarily the same as $B(x)$. ($Hint:$ See Exercise 31.4-4.)

(Omit!)

32.3 String matching with finite automata

32.3-1

> Construct the string-matching automaton for the pattern $P = \text{aabab}$ and illustrate its operation on the text string $T = \text{aaababaabaababaab}$.

$$0 \to 1 \to 2 \to 2 \to 3 \to 4 \to 5 \to 1 \to 2 \to 3 \to 4 \to 2 \to 3 \to 4 \to 5 \to 1 \to 2 \to 3.$$

32.3-2

> Draw a state-transition diagram for a string-matching automaton for the pattern $\text{ababbabbababbababbabb}$ over the alphabet $\sigma = \{a, b\}$.

0	1	0
1	1	2
2	3	0
3	1	4
4	3	5
5	6	0
6	1	7
7	3	8
8	9	0
9	1	10
10	11	0
11	1	12
12	3	13
13	14	0
14	1	15
15	16	8
16	1	17
17	3	18
18	19	0
19	1	20
20	3	21
21	9	0

32.3-3

> We call a pattern P ***nonoverlappable*** if $P_k \sqsupset P_q$ implies $k = 0$ or $k = q$. Describe the state-transition diagram of the string-matching automaton for a nonoverlappable pattern.

$\delta(q, a) \in \{0, 1, q + 1\}$.

32.3-4 ⋆

> Given two patterns P and P', describe how to construct a finite automaton that determines all occurrences of either pattern. Try to minimize the number of states in your automaton.

Combine the common prefix and suffix.

32.3-5

> Given a pattern P containing gap characters (see Exercise 32.1-4), show how to build a finite automaton that can find an occurrence of P in a text T in $O(n)$ matching time, where $n = |T|$.

Split the string with the gap characters, build finite automatons for each substring. When a substring is matched, moved to the next finite automaton.

32.4 The Knuth-Morris-Pratt algorithm

32.4-1

> Compute the prefix function π for the pattern $\text{ababbabbabbababbabb}$.

$$\pi = \{0, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 3, 4, 5, 6, 7, 8\}.$$

32.4-2

> Give an upper bound on the size of $\pi^*[q]$ as a function of q. Give an example to show that your bound is tight.

$|\pi^*[q]| < q$.

32.4-3

> Explain how to determine the occurrences of pattern P in the text T by examining the π function for the string PT (the string of length $m + n$ that is the concatenation of P and T).

$\{q \mid \pi[q] = m \text{ and } q \ge 2m\}$.

32.4-4

> Use an aggregate analysis to show that the running time of KMP-MATCHER is Θ.

The number of $q = q + 1$ is at most n.

32.4-5

> Use a potential function to show that the running time of KMP-MATCHER is $\Theta(n)$.

$\Phi = p$.

32.4-6

> Show how to improve KMP-MATCHER by replacing the occurrence of π in line 7 (but not line 12) by π', where π' is defined recursively for $q = 1, 2, \dots, m - 1$ by the equation
>
> $$\pi'[q] = \begin{cases} 0 & \text{if } \pi[q] = 0, \\ \pi'[\pi[q]] & \text{if } \pi[q] \ne 0 \text{ and } P[\pi[q] + 1] = P[q + 1] \\ \pi[q] & \text{if } \pi[q] \ne 0 \text{ and } P[\pi[q] + 1] \ne P[q + 1]. \end{cases}$$
>
> Explain why the modified algorithm is correct, and explain in what sense this change constitutes an improvement.

If $P[q + 1] \ne T[i]$, then if $P[\pi[q] + q] = P[q + 1] \ne T[i]$, there is no need to compare $P[\pi[q] + q]$ with $T[i]$.

32.4-7

> Give a linear-time algorithm to determine whether a text T is a cyclic rotation of another string T'. For example, arc and car are cyclic rotations of each other.

Find T' in TT.

32.4-8 ⋆

> Give an $O(m|\Sigma|)$-time algorithm for computing the transition function δ for the string-matching automaton corresponding to a given pattern P. (Hint: Prove that $\delta(q, a) = \delta(\pi[q], a)$ if $q = m$ or $P[q + 1] \ne a$.)

Compute the prefix function m times.

Problem 32-1 String matching based on repetition factors

> Let y^i denote the concatenation of string y with itself i times. For example, $(\text{ab})^3 = \text{ababab}$. We say that a string $x \in \Sigma^*$ has ***repetition factor*** r if $x = y^r$ for some string $y \in \Sigma^*$ and some $r > 0$. Let $\rho(x)$ denote the largest r such that x has repetition factor r.
>
> **a.** Give an efficient algorithm that takes as input a pattern $P[1 \dots m]$ and computes the value $\rho(P_i)$ for $i = 1, 2, \dots, m$. What is the running time of your algorithm?

> **b.** For any pattern $P[1 \dots m]$, let $\rho^*(P)$ be defined as $\max_{1 \le i \le m} \rho(P_i)$. Prove that if the pattern P is chosen randomly from the set of all binary strings of length m, then the expected value of $\rho^*(P)$ is $O(1)$.
>
> **c.** Argue that the following string-matching algorithm correctly finds all occurrences of pattern P in a text $T[1 \dots n]$ in time $O(\rho^*(P)n + m)$:
>
> ```
> REPETITION_MATCHER(P, T)
>     m = P.length
>     n = T.length
>     k = 1 + ρ*(P)
>     q = 0
>     s = 0
>     while s <= n - m
>         if T[s + q + 1] == P[q + 1]
>             q = q + 1
>             if q == m
>                 print "Pattern occurs with shift" s
>         if q == m or T[s + q + 1] != P[q + 1]
>             s = s + max(1, ceil(q / k))
>             q = 0
> ```
>
> This algorithm is due to Galil and Seiferas. By extending these ideas greatly, they obtained a linear-time string-matching algorithm that uses only $O(1)$ storage beyond what is required for P and T.

a. Compute π, let $l = m - \pi[m]$, if $m \bmod l = 0$ and for all $p = m - i \cdot l > 0$, $p - \pi[p] = l$, then $\rho(P_i) = m/l$, otherwise $\rho(P_i) = 1$. The running time is $\Theta(n)$.

b.

$$
\begin{aligned}
P(\rho^*(P) \ge 2) &= \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots \approx \frac{2}{3} \\
P(\rho^*(P) \ge 3) &= \frac{1}{4} + \frac{1}{32} + \frac{1}{256} + \cdots \approx \frac{2}{7} \\
P(\rho^*(P) \ge 4) &= \frac{1}{8} + \frac{1}{128} + \frac{1}{2048} + \cdots \approx \frac{2}{15} \\
P(\rho^*(P) = 1) &= \frac{1}{3} \\
P(\rho^*(P) = 2) &= \frac{8}{21} \\
P(\rho^*(P) = 3) &= \frac{16}{105} \\
E[\rho^*(P)] &= 1 \cdot \frac{1}{3} + 2 \cdot \frac{8}{21} + 3 \cdot \frac{16}{105} + \ldots \approx 2.21
\end{aligned}
$$

c.

(Omit!)