Key Concepts

- **Pattern matching** is the process of finding some length-$m$ pattern $P$ in a length-$n$ text $T$.
- The process consists of **guesses** (some position $i$ such that $P$ might start at $T[i]$) and **checks** (some position $j$, where $0 \leq j < m$, where we compare $P[j]$ to $T[i+j]$).
- The **brute-force algorithm** checks every possible guess between $T[0]$ and $T[n-m]$.
  - Time complexity — $O(nm)$
- The **DFA algorithm** builds an automaton that accepts the pattern or goes to a previous state on a mismatch.
  - Preprocessing — $O(m\Sigma)$
  - Time complexity — $O(n)$
  - Space complexity — $O(m\Sigma)$
- The **KMP algorithm** compares left-to-right and shifts based on a failure array.
  - The failure array tells us the length of the largest prefix of $P[0..j]$ that is a suffix of $P[1..j]$, where $j > 0$.
    - Preprocessing — $O(m)$
    - Time complexity — $O(n)$
    - Space complexity — $O(m)$
- The **Boyer–Moore algorithm** compares right-to-left and shifts based on bad characters and good suffixes.
  - The bad character heuristic aligns $P$ with the last occurrence of some mismatched character $P[i] = c$.
  - The good suffix heuristic aligns an already-matched suffix of $P$ with another occurrence of that suffix in $P$.
    - Preprocessing — $O(m + |\Sigma|)$
    - Time complexity — $O(n)$
    - Space complexity — $O(m + |\Sigma|)$
- **Rabin–Karp fingerprinting** hashes length-$m$ windows of $T$ to find occurrences of $P$.
  - Preprocessing — $O(m)$
  - Time complexity — $O(n + m)$ expected-case, $\Theta(nm)$ worst-case
  - Space complexity — $O(1)$
- **Suffix tries** store all suffixes of $T$ in a trie, and **suffix trees** are compressed suffix tries.
- If $P$ occurs in $T$, then $P$ is a prefix of some suffix of $T$, so we can search the trie to perform matching.
  - Preprocessing — $O(n^2)$ na"ively, reducible to $O(n)$ but more complicated
  - Time complexity — $O(m)$
  - Space complexity — $O(n)$

Suggested Readings

- **CLRS**: Chapter 32 (String Matching)
- **Goodrich/Tamassia**: 9.1 (Strings and Pattern Matching Algorithms), 9.2 (Tries)
Practice Questions

CLRS

32.1-1. Show the comparisons the brute force pattern-matching algorithm makes for the pattern \( P = 0001 \) in the text \( T = 00010001010001 \).

32.1-2. Suppose that all characters in the pattern \( P \) are different. Show how to accelerate our brute force pattern-matching algorithm to run in time \( O(n) \) on an \( n \)-character text \( T \).

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 \( P = 26 \)?

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.

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.)

32.3-1. Construct the string-matching automaton for the pattern \( P = \text{aabab} \) and illustrate its operation on the text string \( T = \text{aaababaabaababaab} \).

32.3-2. Draw a state-transition diagram for a string-matching automaton for the pattern \( \text{ababbababbababbababb} \) over the alphabet \( \Sigma = \{a, b\} \).

32.4-1. Compute the KMP failure array for the pattern \( \text{ababbababbababb} \).

Goodrich/Tamassia

R-9.2. Draw a figure illustrating the comparisons done by the brute force pattern-matching algorithm for the case when the text is \( \text{aaabadaabaaa} \) and the pattern is \( \text{aabaaa} \).

R-9.3. Repeat the previous problem for the Boyer–Moore pattern-matching algorithm.

R-9.4. Repeat the previous problem for the Knuth–Morris–Pratt pattern-matching algorithm.

R-9.5. Compute a table representing the last occurrence function used in the Boyer–Moore pattern-matching algorithm for the pattern string

"the quick brown fox jumped over a lazy cat"

assuming the following alphabet (which starts with the space character):

\[ \Sigma = \{\text{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}\} \]

R-9.8. Draw a standard trie for the following set of strings:

\{abab, baba, cccccc, bbaaaa, caa, bbaacc, cbcc, cbca\}.


R-9.10. Draw the compact representation of the suffix trie for the string “minimize minime”.

C-9.5. Say that a pattern \( P \) of length \( m \) is a circular substring of a text \( T \) of length \( n \) if there is an index \( 0 \leq i < m \) such that \( P = T[n - m + i..n - 1] + T[0..i - 1] \); that is, if \( P \) is a substring of \( T \) or \( P \) is equal to the concatenation of a suffix of \( T \) and a prefix of \( T \). Give an \( O(n + m) \) time algorithm for determining whether \( P \) is a circular substring of \( T \).
Additional Practice Questions

1. For each of the following patterns, compute the failure array, the last occurrence function, and the good suffix array.
   
   (a) \( P = ababba \)
   
   (b) \( P = abcdefg \)
   
   (c) \( P = mississippi \)