Special lecture on Information Knowledge Network

-Information retrieval and pattern matching-

The 1st. Preliminary: terms and definitions

Takuya kida
IKN Laboratory,
Division of Computer Science and Information Technology

Today’s contents

What is the pattern matching problem?

Sequential search vs. indexing

Basic terms of text algorithms

Finite automatons

What is the pattern matching problem?

Given a pattern P and a text T,
it is the problem of finding occurrences of P in T.

pattern P: compress

text T:

We introduce a general framework which is suitable to capture an essence of compressed pattern matching according to various dictionary based compressions. The goal is to find all occurrences of a pattern in a text without decompression, which is one of the most active topics in string matching. Our framework includes such compression methods as Lempel-Ziv family, (LZ77, LZSS, LZ78, LZW), byte-pair encoding, and the static dictionary based method. Technically, our pattern matching algorithm extremely extends that for LZW compressed text presented by Amir, Benson and Farach [Amir94]..

Well-known algorithms:

- KMP method (Knuth&Morris&Pratt[1974])
- BM method (Boyer&Moore[1977])
- Karp-Rabin method (Karp&Rabin[1987])

Existence problem and all-occurrences problem

Existence problem:

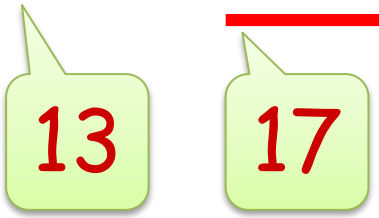
text T : p u r u r u n p u r u n f a m i f a m i f a

pattern P: f a m i f a

All-occurrences problem:

text T : p u r u r u n p u r u n f a m i f a m i f a

pattern P: f a m i f a



When we retrieve a document from a set of documents, it is enough to solve the existence problem. However “pattern matching problem” often means solving the all-occurrences problem.

Two approaches to the problem

Retrieve using sequential search algorithms

advantages:

- ✓ No extra data structures
- ✓ Flexible for data updating

disadvantages:

- ✓ Slow
- ✓ Low scalability

$O(n)$

For small scale group of documents

(e.g. grep of UNIX)

Main topic of this class

Retrieve using index structures

advantages:

- ✓ Very fast
- ✓ High scalability

$O(m \log n)$

disadvantages:

- ✓ Preprocessing for indexing
- ✓ Less flexible for data updating
- ✓ Large memory consumption

For large scale DB

(e.g. namazu, sufary, Google, etc.)

Index structures

(inverted file, suffix tree/array, etc.)

Computational complexity

To evaluate an algorithm, it is necessary to clearly determine the computational complexity of the algorithm

- How much time or how many memory space do we need to calculate for the input data of length n?

big-O notation:

indicates the upper bound of asymptotic complexity
(There is Ω notation to indicate the lower bound)

- Definition: let f and g be functions from an integer to an integer. For some constant C and N, if $f(n) < C \cdot g(n)$ holds for any $n > N$, then we denote $f(n) = O(g(n))$. (we say that f *is order of* g.)
- If f and g are the same order, that is, both $f(n) = O(g(n))$ and $g(n) = O(f(n))$ hold, then we denote $f = \Theta(g)$.
- Let $T(n)$ be a function of the calculation time for the input of length n, and assume that $T(n) = O(g(n))$. This means that "It asymptotically takes only the time proportional to $g(n)$ at most."

e.g. $O(n)$, $O(n \cdot \log n)$

Basic terms for text algorithms

Σ : a finite set of non-empty characters (letters, symbols), called (finite) alphabet.
e.g.: $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots, \mathrm{z}\}$, $\Sigma = \{0,1\}$, $\Sigma = \{0\mathrm{x}00, 0\mathrm{x}01, \ldots, 0\mathrm{xFF}\}$

$x \in \Sigma^*$: a string (word, text), which is a sequence of characters.

$|x|$:length of string x. e.g.: $|\mathrm{aba}| = 3$

ε : the empty string, which is the string whose length is equal to 0, namely $|\varepsilon| = 0$.

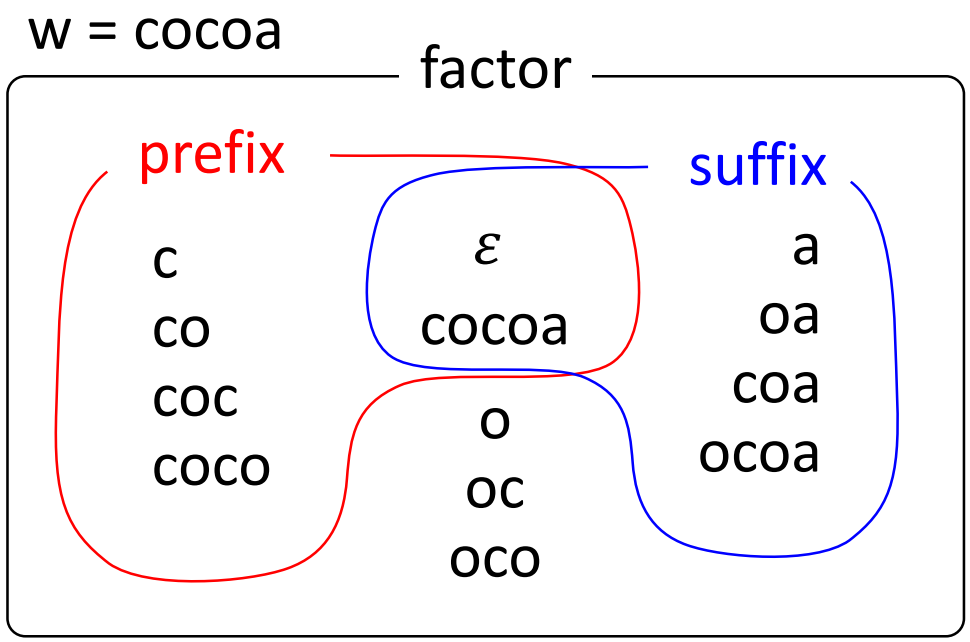
$x[i]$: the i-th character of string x.

$x[i:j]$: the consecutive sequence of characters from the i-th to the j-th of string x, which is called the factor or substring of x.
If $i > j$, we assume $x[i:j] = \varepsilon$ for convenience.
It is often denoted as $x[i..j]$.

x^R : the reversed string $a_k a_{k-1} \ldots a_1$ for string $x = a_1\, a_2 \ldots a_k \in \Sigma^*$.

Prefix, factor, and suffix

Factor $x[1:i]$ is especially called a prefix of x, and $x[i:|x|]$ is especially called a suffix of x.



Note that the difference with a factor!

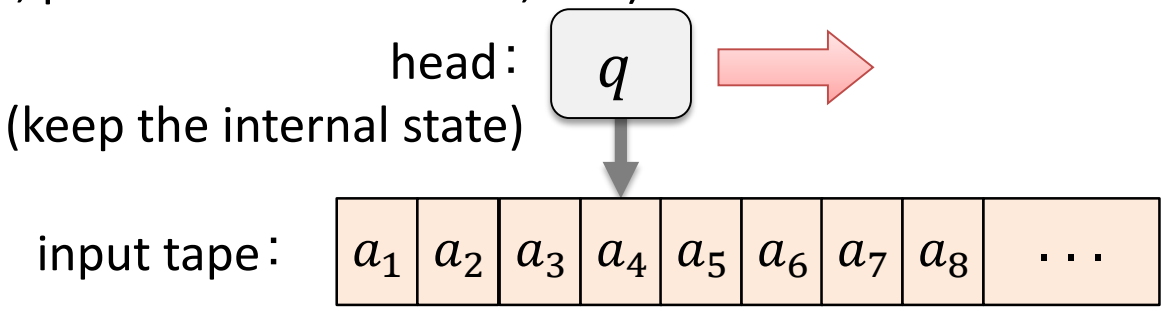
For strings x and y, we say x is a subsequence of y if the string obtained by removing 0 or more characters from y is identical with x.

e.g.: $x = \text{abba}$ is a subsequence of $y = \text{aaababaab}$.

Finite Automaton

fundamental concept on computer science!

It is automatic machine (automata) that determines if the input string is acceptable or not. It repeatedly changes its internal state reading characters from the input one by one. There are several variations according to the definition of the state transitions and the use of auxiliary spaces. (e.g. non-deterministic automaton, sequential machine, pushdown automaton, etc.)



It has the ability to define a (formal) language; it is often used for lexical analysis of strings.

It deeply relates to the pattern matching

References:"Automaton and computability," S. Arikawa and S. Miyano, BAIFU-KAN (written in Japanese)
"Formal language and automaton," Written by E. Moriya, SAIENSU-SHA (written in Japanese)

Deterministic finite automaton (DFA)

DFA M is a quintuple (5-tuple) $(K, \Sigma, q_0, \delta, F)$:

K : a set of internal states $\{q_0, q_1, \dots, q_n\}$ of M.

Σ : an alphabet (a set of characters) $\{a_0, a_1, \dots, a_k\}$.

q_0 : the initial state of M.

δ : a function $K \times \Sigma \to K$ that returns the next state when reading a character from the input. It is called a transition function.

F : a set of accept states (or final state). $F \subseteq K$.

We extend the domain of δ from $K \times \Sigma$ to $K \times \Sigma^*$ as follows:

$$\delta(q, \varepsilon) = q \quad (q \in K),$$
$$\delta(q, ax) = \delta(\delta(q, a), x). \quad (q \in K, a \in \Sigma, x \in \Sigma^*)$$

Then, $\delta(q_0, w)$ indicates the state when string w is input.

We say that M accepts w if $\delta(q_0, w) = p \in F$.

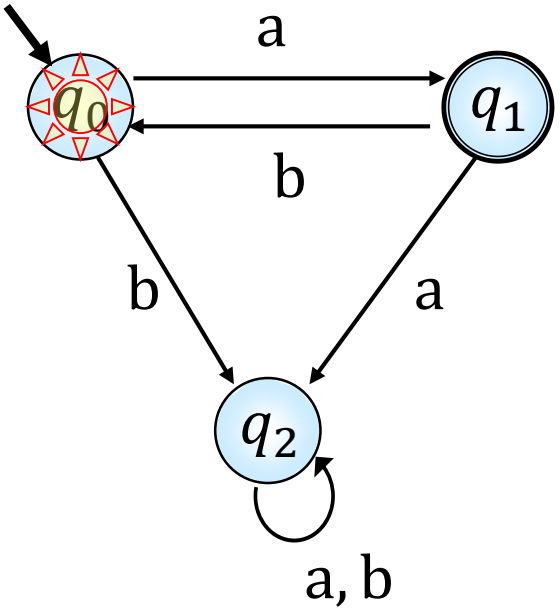
The set $L(M) = \{w \mid \delta(q_0, w) \in F\}$ of all strings that M accepts is called the language accepted by finite automaton M.

An example of DFA

State transition

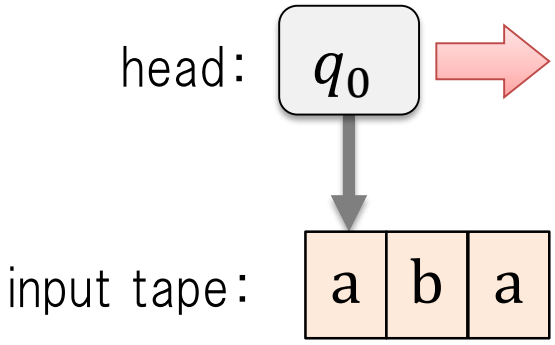
$$\begin{aligned}\delta(q_0, \mathrm{aba}) &= \delta(\delta(q_0, \mathrm{a}), \mathrm{ba}) \\ &= \delta(q_1, \mathrm{ba}) \\ &= \delta(\delta(q_1, \mathrm{b}), \mathrm{a}) \\ &= \delta(q_0, \mathrm{a}) \\ &= q_1 \in F\end{aligned}$$

State transition diagram



Language that M accepts

$$L(M) = \{\mathrm{a}(\mathrm{ba})^n \mid n \geq 0\}$$



Nondeterministic finite automaton (NFA)

NFA M is a quintuple $(K, \Sigma, Q_0, \delta, F)$:

K : a set of internal states $\{q_0, q_1, \ldots, q_n\}$ of M.

Σ : an alphabet (a set of characters) $\{a_0, a_1, \ldots, a_k\}$.

Q_0 : the initial state of M. $Q_0 \subset K$.

δ : a transition function $K \times \Sigma \to 2^K$ of M.

F : a set of accept states. $F \subseteq K$.

The destination of each transition isn't unique.

Some states can be active.

We extend the domain of δ from $K \times \Sigma$ to $K \times \Sigma^*$ as follows:

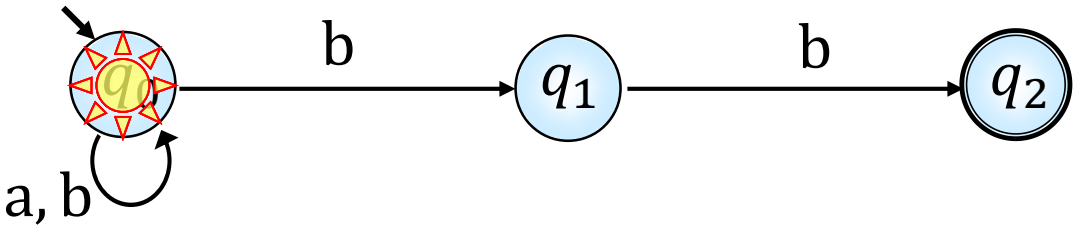
$$\delta(q, \varepsilon) = \{q\},$$
$$\delta(q, ax) = \cup_{p \in \delta(q,a)} \delta(p, x). \quad (q \in K, a \in \Sigma, x \in \Sigma^*)$$

Moreover we extends it to $2^K \times \Sigma^*$:

$$\delta(S, x) = \cup_{q \in S} \delta(q, x).$$

For $x \in \Sigma^*$, we say that M accepts x if $\delta(Q_0, x) \cap F \neq \phi$.

An example of NFA



State diagram of NFA

For string abb, $\delta(q_0, \text{abb}) = \delta(q_0, \text{bb})$

$= \delta(q_0, \text{b}) \cup \delta(q_1, \text{b})$

$= \{q_0\} \cup \{q_1\} \cup \{q_2\}$

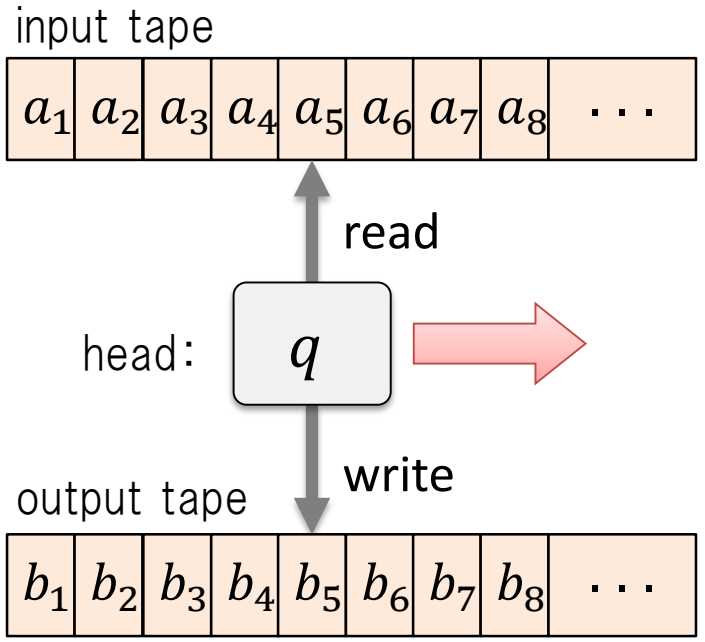
$= \{q_0, q_1, q_2\}$

Then, abb $\in L(M)$ since $q_2 \in F$.

Sequential machine

Sequential machine M, a variation of DFA that sequentially outputs to an input, is a sextuple (6-tuple) $(K, \Sigma, \Delta, q_0, \delta, \lambda)$:

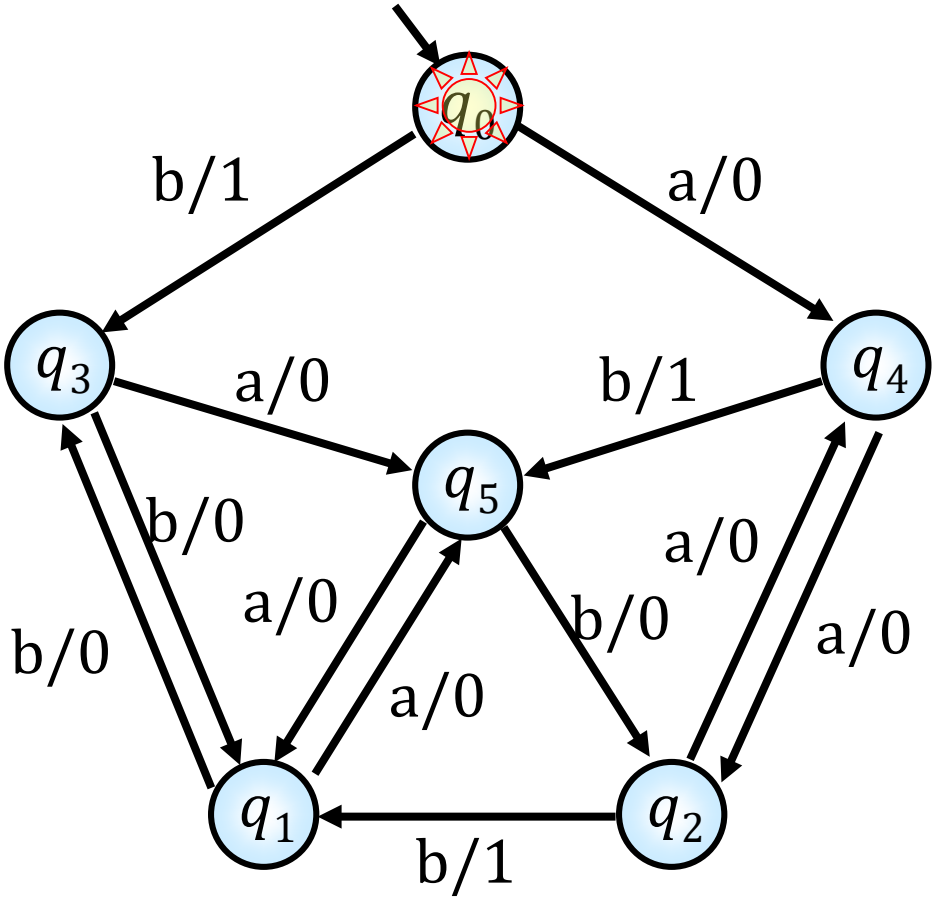
- K : a set of internal states $\{q_0, q_1, \ldots, q_n\}$
- Σ : an alphabet of the input $\{a_0, a_1, \ldots, a_k\}$
- Δ : an alphabet of the output $\{b_0, b_1, \ldots, b_\ell\}$
- q_0 : the initial state.
- δ : the transition function $K \times \Sigma \to K$.
- λ : the output function $K \times \Sigma \to \Delta$.



We extend the domain of δ to $K \times \Sigma^*$ in a similar way of DFA. Moreover we extends the domain of λ to that from $K \times \Sigma^*$ to Δ^* as follows:

$$\lambda(q, \varepsilon) = q, \quad (q \in K)$$
$$\lambda(q, ax) = \lambda(q, a)\lambda(\delta(q, a), x). \quad (q \in K, a \in \Sigma, x \in \Sigma^*)$$

An example of sequential machine



state diagram

state transition and outputs

$$\lambda(q_0, \mathrm{abb})$$

$$= \lambda(q_0, \mathrm{a})\lambda(\delta(q_0, \mathrm{a}), \mathrm{bb})$$

$$= 0\lambda(q_4, \mathrm{bb})$$

$$= 0\lambda(q_4, \mathrm{b})\lambda(\delta(q_4, \mathrm{b}), \mathrm{b})$$

$$= 01\lambda(q_5, \mathrm{b})$$

$$= 010$$

it's a kind of translator!

Summary

What is the pattern matching problem?
- The problem of finding the occurrences of pattern P in text T.
- There are the existence problem and all-occurrence problem.

The difference between sequential search and indexing
- The former is usually slower than the latter. However, the former doesn't need any auxiliary index structure.

Basic terms and notations
- Computational complexity: big-O notation
- Alphabet, string, prefix, factor, and suffix

Finite automaton
- DFA defines a language.
- NFA has multiple current states.
- Sequential machine outputs for each input character.