北海道大学
Hokkaido University

Lecture on Information Knowledge Network "Information retrieval and pattern matching"

Laboratory of Information Knowledge Network,
Division of Computer Science,
Graduate School of Information Science and Technology,
Hokkaido University

Takuya KIDA

The 1st Preliminary: terms and definitions

What is the pattern matching problem?
Basic terms of text algorithms
About finite automaton
Difference with text retrieval using index data structure

What is the pattern matching problem?

- Problem of finding occurrences of pattern P in text T
  Well-known algorithms:
  - KMP method (Knuth&Morris&Pratt 1974)
  - BM method (Boyer&Moore 1977)
  - Karp-Rabin method (Karp&Rabin 1987)

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].

Existence problem and all-occurrences problem

Existence problem

Text T: tekumakumayakontekumakumayakon

Pattern P: kumakuma

Yes!

All-occurrences problem

Text T: tekumakumayakontekumakumayakon

Pattern P: kumakuma

3

18

Although it is enough for us to solve the existence problem when we retrieve a document from a set of documents, "solving the pattern matching problem" often means solving the all-occurrences problem.

Basic terms (computational complexity)

- To clarify the quality of an algorithm, we have to judge the computational complexity
  - How much time and memory space do we need for the calculation to the input data of length n?
- Big-O notation
  - 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 write $f(n) = O(g(n))$. (f is said to be "order of g")
  - If f and g are the same order, that is, $f(n)=O(g(n))$ and $g(n)=O(f(n))$, then we write $f=\Theta(g)$.
  - 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."

indicates the upper bound of asymptotic complexity
(There is Ω notation, which indicates the lower bound)

Example: $O(n)$, $O(n\cdot \log n)$

Basic terms (alphabet and string)

- Definition of terms
  - Σ: a finite set of non-empty characters. (The set is called alphabet)
    - Example: Σ={a,b,c,…}, Σ={0,1}, Σ={0x00, 0x01, 0x02,…, 0xFF}
  - $x \in \Sigma^*$: a string, a word, or a text
    - |x|: length of string x. Example: |aba|=3.
    - ε : the string whose length is equal to 0 is called the empty string. That is, $|\varepsilon|=0$.
  - x[i]: the character of i-th position of string x.
  - x[i.. j]: the consecutive sequence of characters from i to j of string x
    - We assume x[i.. j]=ε for i>j for convenience.
    - x[1..i] is especially called a prefix of x.
    - x[i..|x|] is especially called a suffix of x.
  - 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 identified with x.
    - Example: X=abba is a subsequence of y=aaababaab.
  - For $a_1a_2\cdots a_k \in \Sigma^*$, we denote the reversed string $a_ka_{k-1}\cdots a_1$ by x^R.

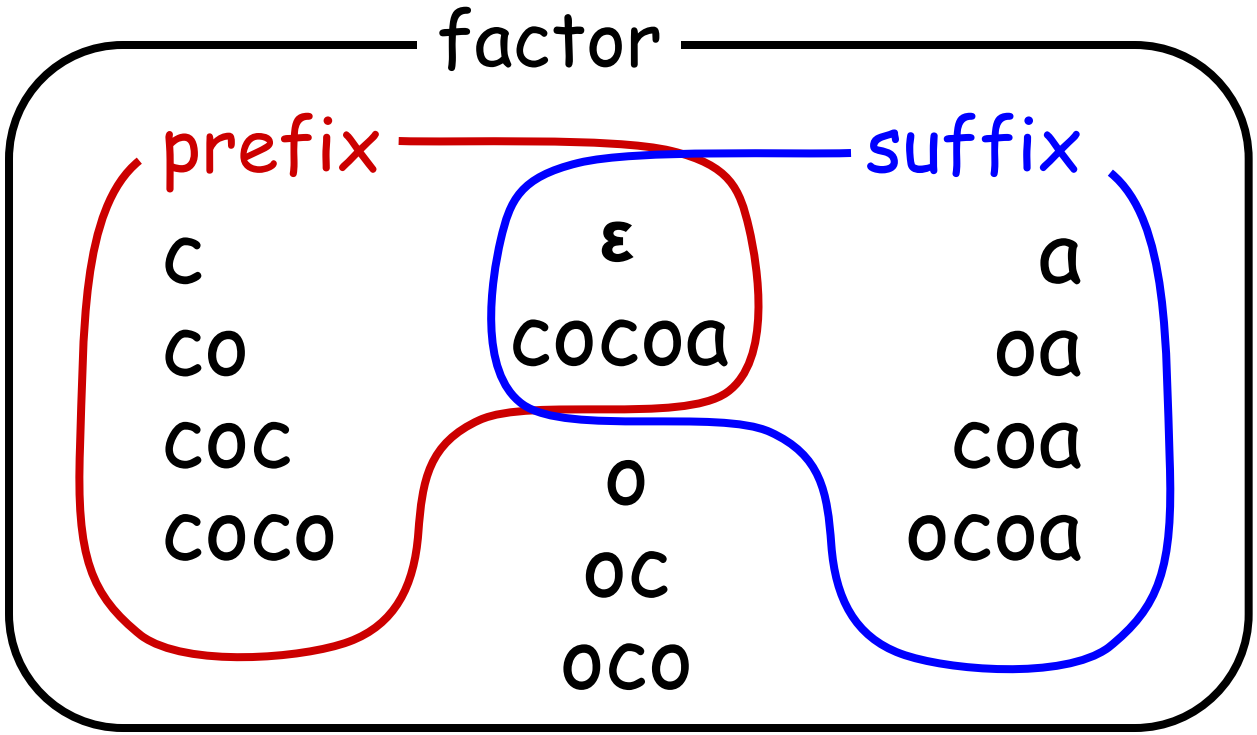
Character a∈ ∑ is also called a letter or a symbol

It is called factor, substring, or subword of x.

Note that the difference with a factor!

Example of prefix, factor, and suffix

w = cocoa



exercise 1: Enumerate all the prefix of w=ROYCE.
exercise 2: Enumerate all the factor of w=ABABC.

Finite automaton

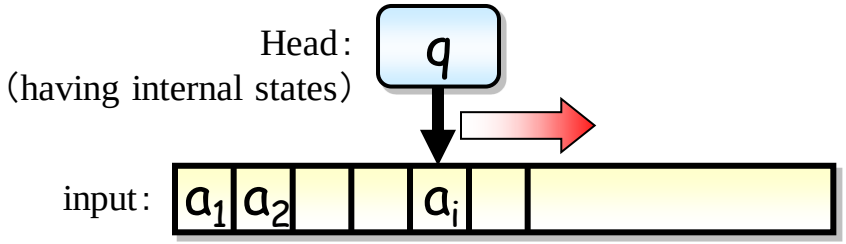
- What is a finite automaton?
  - Automatic machine (automata) where the output is determined by an input and its internal states.
  - The number of states is finite.
  - There are several variations according to the definition of its state transitions and output, and the use of the auxiliary spaces.
    - For example, non-deterministic automaton, sequential machine, pushdown automaton, and etc.
  - It has the ability to define (computer) languages; it is often used for lexical analysis of strings.
    - It is one of the most fundamental concepts that appears in all computer science fields!
    - It is used also for software design (state transition diagram of UML)!
    - Of course, it deeply relates to the pattern matching problem!

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)

Definition of deterministic finite automaton

- (Deterministic) Finite automaton M = (K, Σ, δ, q0, F)
  - K = {q0, q1, …, qn}: a set of states,
  - Σ = {a, b, …, c}: a set of characters (alphabet),
  - q_0 : initial state,
  - δ: transition function $K \times \Sigma \rightarrow K$
  - F: a set of accept states (which is a subset of K)

- About the transition function
  - We extend it as follows:
    - δ(q, e) = q (q ∈ K)
    - δ(q, ax) =δ(δ(q, a), x) (q∈K, a∈Σ, x∈Σ*)
  - For string w, $\delta(q_0, w)$ indicates the state when w is input.
  - We say that M accepts w if and only if $\delta(q_0, w) = p \in F$.
  - The set L(M)={w|δ(q0, w)∈F} of all strings that M accepts is called the language accepted by finite automaton M.



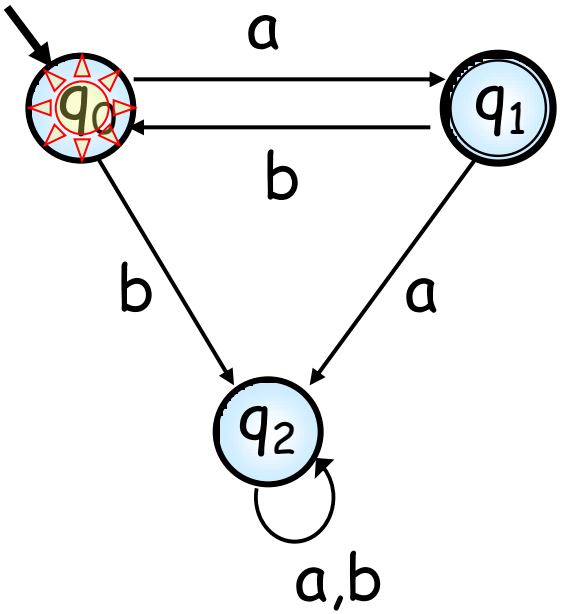
Example of deterministic finite automaton

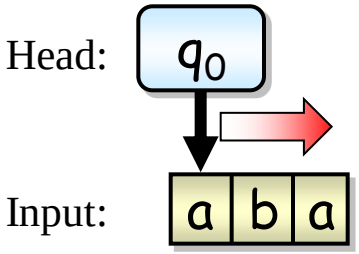
State transition

$$\delta(q_0, aba) = \delta(\delta(q_0, a), ba)$$
$$= \delta(q_1, ba)$$
$$= \delta(d(q_1, b), a)$$
$$= \delta(q_0, a)$$
$$= q_1 \in F$$

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

State transition diagram





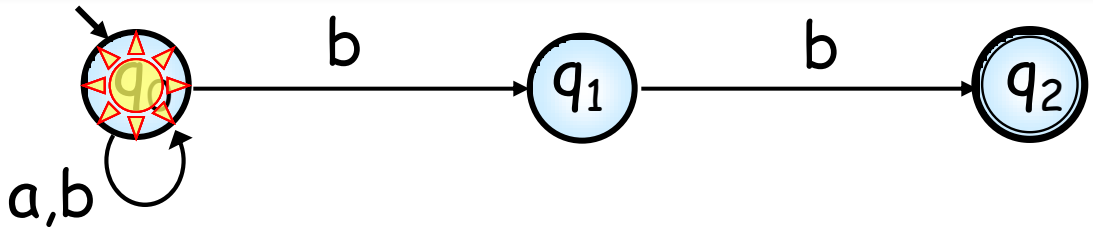
Definition of nondeterministic finite automaton

- Nondeterministic finite automaton $M=(K, \Sigma, \delta, Q_0, F)$
  - $K = \{q_0, q_1, \ldots, q_n\}$: a set of states,
  - $\Sigma = \{a, b, \ldots, c\}$: a set of characters (alphabet),
  - $Q_0 \subset K$: a set of initial states,
  - δ: transition function $K \times \Sigma \rightarrow 2K$
  - F: a set of accept states (which is a subset of K)
- About the transition function:
  - We extend the domain from $K \times \Sigma$ to $K \times \Sigma^*$ as follows:
    - $\delta(q, e) = \{q\}$
    - $\delta(q, ax) = \bigcup_{p \in \delta(q, a)} \delta(p, x)$ $(q \in K, a \in \Sigma, x \in \Sigma^*)$
  - Moreover, we extend it to $2K \times \Sigma^*$.
    - $\delta(S, x) = \bigcup_{q \in S} \delta(q, x)$
  - For $x \in \Sigma^*$, we say that M accepts x when $\delta(Q0, x) \cap F \neq \Phi$.

In other words, destination of each transition isn't unique!

There exist several current states!

Example of nondeterministic automaton



State diagram of nondeterministic finite automaton

For abb, for example, $\delta(q_0, abb) = \delta(q_0, bb)$

$= \delta(q_0,b) \cup \delta(q_1,b)$

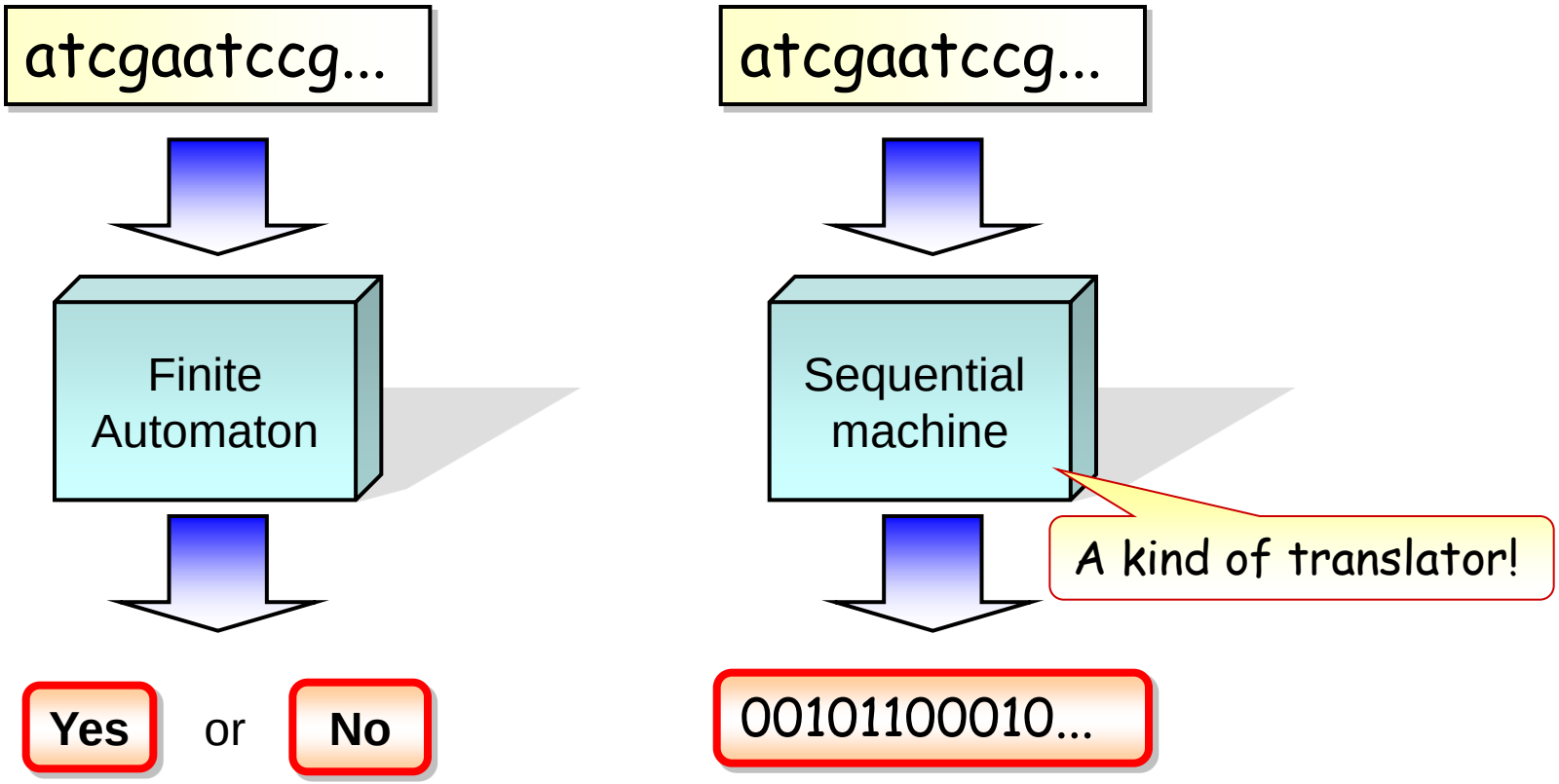
$= \{q_0\} \cup \{q_1\} \cup \{q_2\}$

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

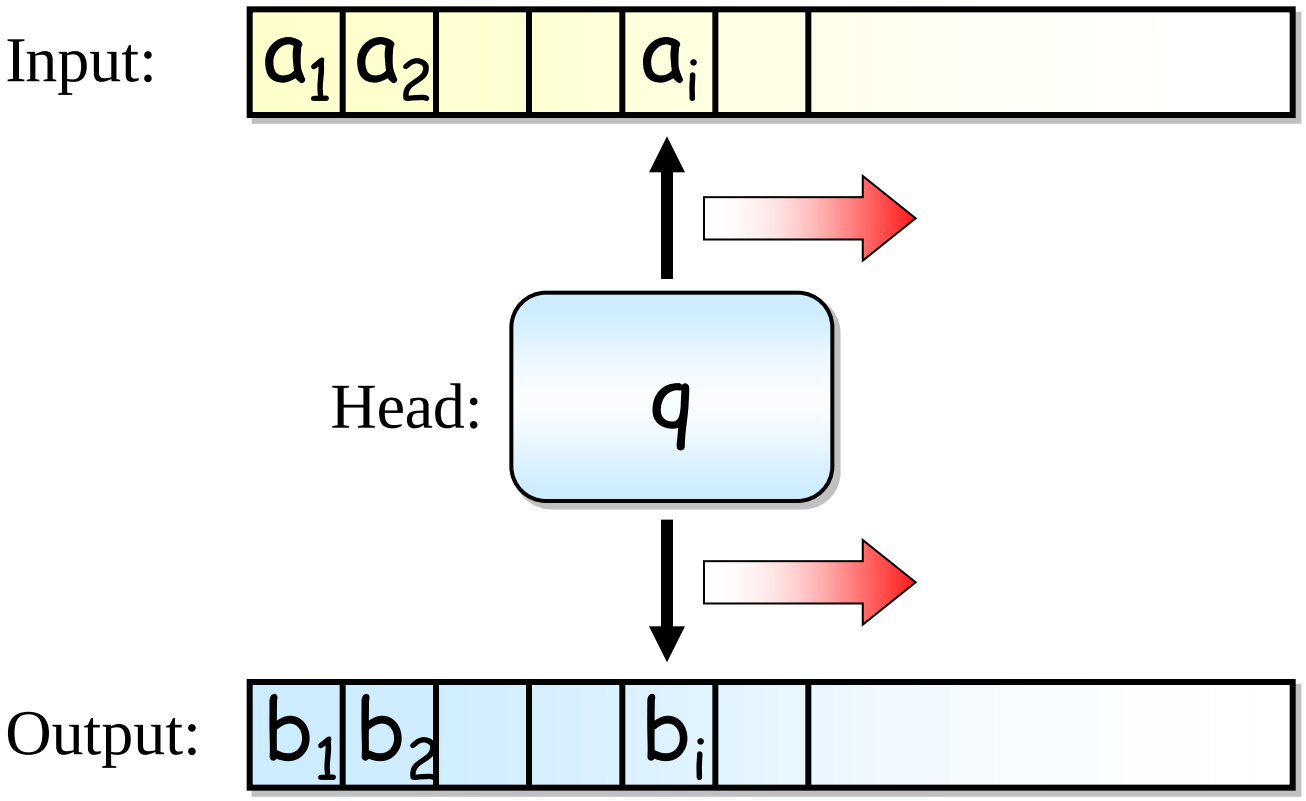
Then, $abb \in L(M)$ since $q_2 \in F$.

Sequential machine

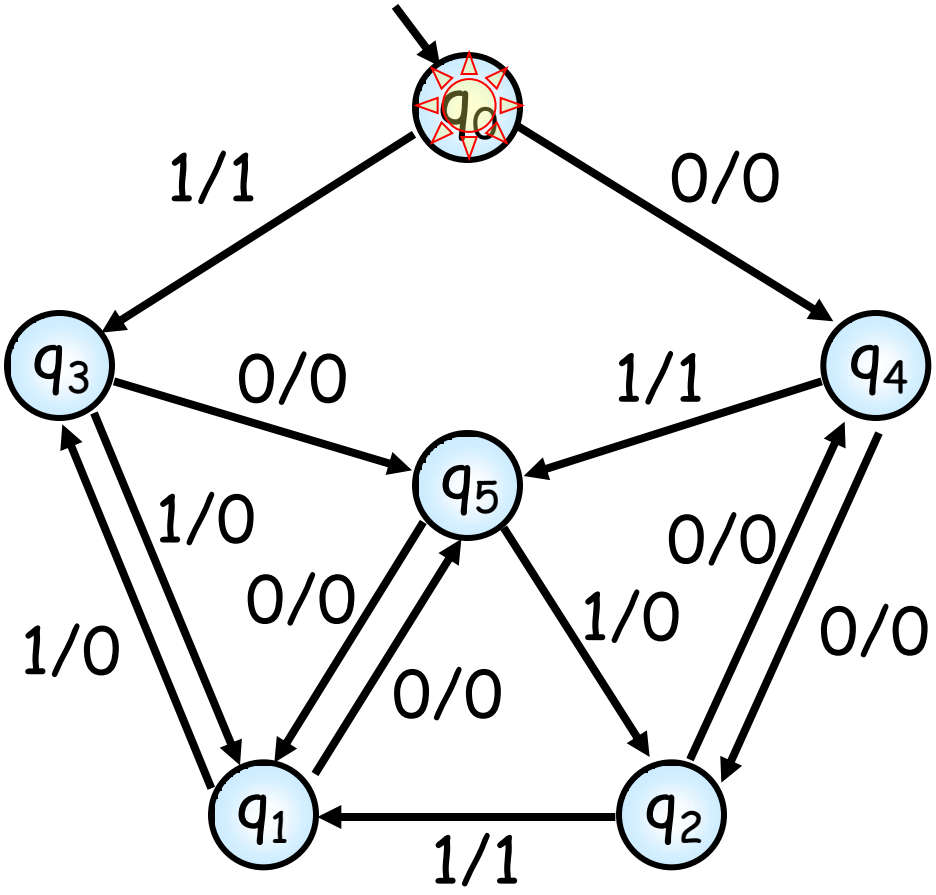
What is a sequential machine?



Conceptual diagram of sequential machine



Example of sequential machine



$\lambda(q_0, 011)$

$= \lambda(q_0, 0)\lambda(\delta(q_0,0), 11)$

$= 0\lambda(q_4, 11)$

$= 0\lambda(q_4, 1)\lambda(\delta(q_4,1), 1)$

$= 01\lambda(q_5, 1)$

$= 010$

Difference with text retrieval using index data structure

Text retrieval by doing pattern matching

- Merit
  - ✓ No extra data structure
  - ✓ Flexible about data updating
- Weak point
  - ✓ Slow
  - ✓ Low scalability

O(n)

For small-scale group of documents
(example: grep of UNIX)

Text retrieval by using index data structure

- Merit
  - ✓ Very fast
  - ✓ High scalability
- Weak point
  - ✓ To construct the index is needed
  - ✓ Little flexibility of updating
  - ✓ Space for the index is necessary

O(m log n)

For large-scale DB
(example: Namazu, sufary, mg, Google)

There exists
a large-scale full-text search DB system
based on pattern matching!

Summary (the 1st)

- What is the pattern matching problem?
  - Problem of finding the occurrences of pattern P included in text T
  - There are the existence problem and the all-occurrence problem.
- Basic terms of text algorithm
  - Notation of computational complexity: Big-O notation
  - Alphabet, string, prefix, factor, and suffix
- Finite automaton
  - Deterministic finite automaton: it can define computer languages.
  - Nondeterministic finite automaton: There are some existing states and destinations of each transition.
  - Sequential machine: it outputs for each input character.
- Difference with text retrieval using index data structure
  - Although the text retrieval by pattern matching is usually considered that it is slower than the retrieval using index data structure, the former has some good aspects!