Finite Automata Pattern Matching

Many string-matching algorithms build a finite automaton—a simple machine for processing information—that scans the text string T for all occurrences of the pattern P. This topic presents a method for building such an automaton.

Overview

A **finite automaton M**, is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$ where

- Q is a finite set of *states*
- $q_0 \in Q$ is the *start state*
- $A \subseteq Q$ is a distinguished set of *accepting states*
- Σ is a finite *input alphabet*
- δ is a function from $Q \times \Sigma$ into Q, called the *transition function* of M.

Algorithm

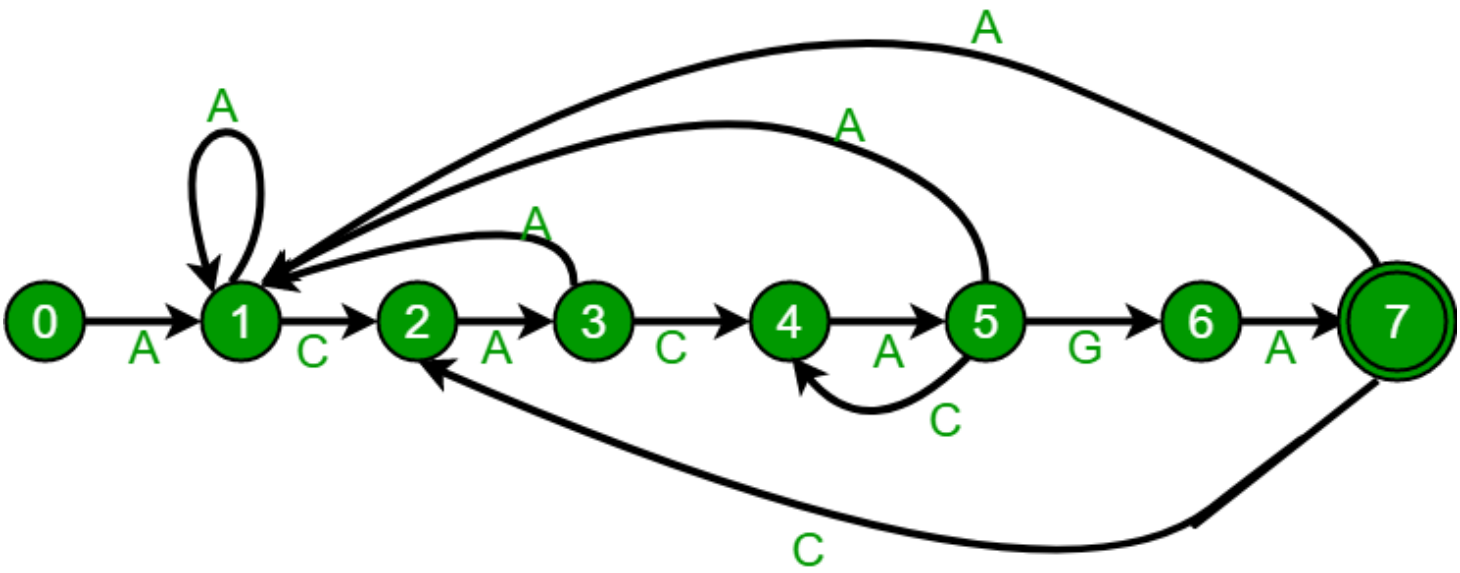
The Finite Automaton(FA) begins in state q_0 and reads the characters of its input string one at a time. If the automaton is in state q and reads input character a, it moves ("makes a transition") from state q to state $\delta(q, a)$. Whenever its current state q is a member of A, the machine M has *accepted* the string read so far. An input that is not accepted is *rejected*.

A finite automaton M induces a function ϕ, called the *final-state function*, from Σ^* to Q such that $\phi(w)$ is the state M ends up in after scanning the string w. Thus, M accepts a string w if and only if $\phi(w) \in A$.

String-matching automata

For a given pattern P, we construct a string-matching automaton in a preprocessing step before using it to search the text string. Number of states in FA will be $m + 1$ where m is length of the pattern. The main logic to construct FA is to get the next state from the current state for every possible character. Given a character x and a state k, we can get the next state by considering the string $pat[0..k-1]\ x$ which is basically concatenation of pattern characters $pat[0], pat[1]\ \ldots\ pat[k-1]$ and the character x.

The idea is to get length of the longest prefix of the given pattern such that the prefix is also suffix of $pat[0..k-1]\ x$. *The value of length gives us the next state*.



For example, for input pattern `ACACAGA`, to get the next state from `state 5` and character `C` in the above diagram. We need to consider the string, $pat[0\ ..\ 4]C$ which is `ACACAC`. The length of the longest *prefix of the pattern* such that the prefix is equal to *suffix of* `ACACAC` is 4 `ACAC`. So the next state (from `state 5`) is `state 4` for character `C`.

Time and Space Complexity

The running time of COMPUTE-TRANSITION-FUNCTION is $O(m^3|\Sigma|)$ where m is length of the pattern. With some improved procedure for computing Transition Function, we can find all occurrences of a length m pattern in a length n text over an alphabet Σ with $O(m|\Sigma|)$ preprocessing time and $\Theta(n)$ matching time.

References

- DFA Pattern matching geeksforgeeks