



1 Failure links automaton

The **Aho-Corasick** algorithm allows efficient searching for multiple patterns in non-indexed strings. The first step in the **Aho-Corasick** algorithm is to construct the **failure links automaton** for the patterns that we need to search for. The following animated figure explains how to construct the failure links automaton for the patterns { **AGA**, **AA**, **AAG**, **GAAG**, **TCG** }. Each mouse click on the following animation moves one step:

To construct the failure links automaton of the patterns that we need to search for, we start by constructing the trie which corresponds to these patterns. Then for each node p_i other than the root node p_0 , construct a failure link that starts from node p_i and goes to another node as follows: Consider the string s_i labeling edges on the path from the root node p_0 to node p_i . The failure link starting at node p_i must end at node p_j , where p_j is the location of the string s_j in the trie, where

s_j is the longest proper suffix of s_i that has a location in the trie (it is possible to traverse the trie starting from root using s_j). If no such suffix exists, the failure link starting from node p_i must end at the root node p_0 .

For example, to find the destination of the failure link starting from node p_8 , consider the string GAA which starts from the root node p_0 and ends at node p_5 . The proper suffixes of that string, ordered from longest to shortest, are: (AA, A) . The longest proper suffix of these 2 suffixes that also has a location in trie starting from p_0 is AA . Since starting from p_0 the suffix AA ends at trie node p_4 , the failure link starting from node p_8 must end at node p_4 .

2 Aho-Corasick algorithm

The following animated figure is an example of applying the **Aho-Corasick** algorithm to search for the patterns $\{AGA, AA, AAG, GAAG, TCG\}$ inside the text $GAACAAGTGAAGTGAGAAGAAGT$. Each mouse click on the following animation moves one step:

To search for a set of patterns P in text t of size $|t|$, the **Aho-Corasick** algorithm proceeds as follows: Construct the failure links automaton of P . Construct a simple automaton for text t

without failure links. The algorithm keeps track of two nodes: a node in the automaton of t and another node in the automaton of P . The algorithm starts from the state (t_0, p_0) which are the initial node in the t and the root of p automata. The algorithm terminates when it reaches the state $(t_{|t|}, p_0)$.

Suppose the algorithm is at state (t_i, p_j) :

- If the edges $t_i \rightarrow t_{i+1}$ and $p_j \rightarrow p_{j+1}$ have equal labeling characters, go to (t_{i+1}, p_{j+1}) . Then report an occurrence of the pattern ending at node p_{j+1} (if exists) starting from t_{i-plen} where $plen$ is the size of the pattern ending at p_{j+1} . Also, report all patterns ending at the nodes on the failure links path starting from p_{j+1} and ending at p_0 .
- Otherwise:
 - If $j = 0$, go to (t_{i+1}, p_j) .
 - Otherwise, go to $(t_i, f[p_j])$, where $f[p_j]$ is the destination node of the failure link of node p_j .

The complexity of the **Aho-Corasick** algorithm is $O(|t| + k|p|)$ (plus the number of reported pattern occurrences), where k is the number of patterns and $|p|$ is the average size of one pattern. This is asymptotically better than applying the **KMP** algorithm k times on the k patterns which has the complexity $O(k \times (|t| + |p|)) = O(k|t| + k|p|)$. For official proofs regarding the correctness and complexity of the **Aho-Corasick** algorithm, consult the reference text book.