Aho-Corasick

Lawrence Wang

April 29 2016

1 Introduction

Consider the multiple string matching problem: given a set of K pattern strings and a text string of length N, find all occurrences of patterns within the text. If we repeatedly use a single string matching algorithm, such as KMP, we can obtain the answer in $O(K(N+M))$, which is too slow. The goal of this lecture is to develop an algorithm to solve this in time linear to the total length of the patter strings M, the length of the text N, and the total number of matches.

2 FSA Construction

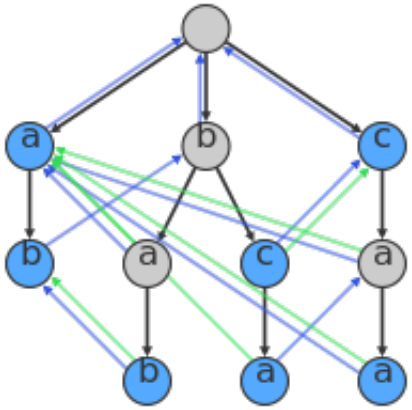
We want to create a Finite State Machine that will allow us to prevent backtracking while searching through the text. We can think of a FSA as a collection of states, where each state corresponds to a position in our pattern strings. Each state also stores two pointers to two other states, corresponding to a match and mismatch.

To begin, let's first put all of our strings in a trie. This is useful in a number of ways. Any states that occur in multiple strings of our set are nicely represented as a single state. Additionally, every pointer from a node to its child represents a state transition for a match. Now all we still need to do is to compute the pointers for each state in the case a mismatch occurs, which are called suffix links.

Suppose that we are at a state i which corresponds to character c. If we find a match, then we can move to i's child accordingly. However, if a mismatch occurs, we need to transition to some previous state that corresponds to the longest suffix of i since all of the suffixes of i may still result in a match.

We can accomplish this easily with the following observation: the longest suffix of state i must be a suffix of i's parent with c appended. This means that we can we can repeatedly traverse i's parent's suffix links until we reach a state that has a child c. The child of the first such state that we encounter will be the longest suffix of i. If no such state is found, the i's suffix link points to the root.

Finally, to construct all of the suffix links we just run a BFS starting at the root and execute the operations above whenever we visit a state. The BFS ensures that when we try to create a state i's suffix link, all states of height less than i's height have already been created. We can accomplish this in time linear to the total number of states in the trie.



3 Search Algorithm

Now that the finite state machine is complete, we can search a text string as follows. We begin at the root and read in a character c. If the current state has a child c, then we follow that edge and read in the next character. Otherwise, we follow the current state's suffix link and do not read in the next character. We repeatedly travel the suffix links when we find a mismatch until we are at the root and we still have a mismatch. In this case, we read the next character and repeat the above steps.

Whenever we arrive at a state that corresponds to the end of a string in the set, we have found a match. Similarly, if we do not arrive at such a state after reading through all of the text string, then no matches were found. There is one caveat however. Since we are searching for matches with multiple strings, a match with one string might also imply a match with another string that has the same suffix. Although these two strings are represented by different states, these states are connected by suffix links. Consequently, whenever we find a match we must travel along suffix links to the root, taking note of any state that corresponds to the end of a string.

We can show that this algorithm runs in linear time. Whenever we read in a character and find a match, we are increasing the current state from height h to $h + 1$. We may later transition from height $h + 1$ to h again by following suffix links towards the root. As a result, the total number of operations is bounded by $2N$, where N is the length of the text string. The only exception to this is when we find a match, which causes to traverse suffix links back to the root.

Thus, our algorithm runs in the $O(N + M + K)$, where N is the length of the text, M is the total length of all patterns, and K is the total number of matches. Note that K may be quadratic, but this cannot be helped.

4 Problems

1. USACO 2015 February Contest, Gold Problem 2. Censoring
2. UVa - 10679. I Love Strings!!