14.1 Finishing up finite automata

Recall that for an alphabet $A$, we can define a language $L \subseteq A^*$. DFAs accept languages. Nondeterministic finite automata also accept languages. Regular expressions denote languages.

**Theorem 14.1** DFAs, NDFAs, regular expressions, regular languages, and regular grammars are all equivalent.

Nondeterministic finite automata have the state transition function $\delta : S \times A \rightarrow 2^S$, where $2^S$ is the size of the set of all $R \subseteq S$.

**Definition 14.2** $A_1 \equiv A_2$ if $L(A_1) = L(A_2)$ for $A_1, A_2$ automata.

14.2 Suffix trees

We have learned how to find a single pattern $p$ in a text $t$. What if we want to find several patterns $p_1, p_2, \ldots, p_n$ in a text $t$? Let’s first examine a simpler version of this problem: Given a list of numbers $L = (\alpha_1, \ldots, \alpha_n)$ and a number $\beta$, determine whether $\beta$ is in $L$. It can be shown that $n$ comparisons are both necessary and sufficient to either find $\beta$ in $L$ or determine that $\beta$ is not in $L$.

Moving up a level in complexity, what if we wanted to find $m$ numbers $\beta_1, \ldots, \beta_m$ in the list $L$? Using the naïve approach, we need $mn$ comparisons. Can we do better? In 1945, John von Neumann invented the merge sort algorithm, which sorts an array of $n$ numbers in $O(n \log n)$ time. If we first sort $L$, then we can perform a binary search on the list $L$. Finding $\beta_1$ in a sorted $L$ only requires $O(\log n)$ time. So altogether we need $O((m + n) \log n)$ to find $\beta_1, \ldots, \beta_m$ in $L$, beating the naïve approach.

The key insight is the following: Sorting was a preprocessing step that allowed future queries to be performed more efficiently.

14.2.1 Online search

Returning to pattern matching on strings, using suffix trees (which we’ll define shortly) gives us the surprising result that we can determine whether a pattern occurs in a text as soon as we finish reading the pattern.
Let $\mathcal{A}$ be the alphabet (e.g. \{A, C, G, T\} for DNA). We will make use of the symbol $\$ to denote the end of a string, with the property that $\notin \mathcal{A}$. For example, if $\mathcal{A} = \{a, b\}$, then we might have a string $x = abaab$, and $x\$ = $abaab\$, where $\$ marks the end of the string. There are $|x\| = 6$ suffixes of $x\$. To see this, let $\text{suf}_i$ denote the suffix of $x\$ starting at position $i$ (1-indexed). Then,

\[
\begin{align*}
\text{suf}_1 &= abaab\$ \\
\text{suf}_2 &= baab\$ \\
\text{suf}_3 &= aab\$ \\
\text{suf}_4 &= ab\$ \\
\text{suf}_5 &= b\$ \\
\text{suf}_6 &= \$
\end{align*}
\]

Now think “finite automata.” We can draw the following tree picture.

Think about how you would determine whether a pattern occurs in the text $x$ using the tree. Let’s make this tree a DFA. Recall that a DFA is defined by $(S, \mathcal{A}', \delta, s_0, F)$. In this case, we have

\[
\begin{align*}
S &= \text{all nodes} \\
\mathcal{A}' &= \{a, b, \$\} \\
s_0 &= \text{root} \\
F &= S \setminus \{\text{sink state}\} \\
\delta &= \text{“all edges and all missing edges go to a sink state”}
\end{align*}
\]

Observe that the language accepted by this DFA is precisely $L = \text{the set of substrings of } x$. We can determine whether a pattern is a substring of $x$ by feeding the pattern into the DFA and seeing if it is accepted. This only requires as much time as it takes to read the pattern, \textit{independent of the length} of the text $x$! Preprocessing the text allows us to check whether a pattern occurs in that text much faster. Furthermore,
it can be shown that the “suffix tree” above can be constructed in $O(n)$ time where $n$ is the length of the text. (Take CS182 if you want to find out how!) The space cost of storing the suffix tree is $O(n^2)$.

To formalize a little, denote the raw string by $x$, the string with the dollar sign by $x\$, and the expanded suffix tree by $T_x$. (We call it the expanded suffix tree because we will see a “compressed” suffix tree later on.) The tree $T_x$ is a search tree that collects all the suffixes of $x\$.

**Definition 14.3** Let $n = |x\|$ with $x \in A^{n-1}$.

1. $T_x$ has $n$ leaves labeled $1, \ldots, n$.
2. Each edge is labeled with a symbol from $A \cup \{\$\}. For every $i$, with $1 \leq i \leq n$, the concatenation of the labels on the path from the root of $T_x$ to leaf $i$ is precisely the suffix $suf_i = x_i x_{i+1} \ldots x_{n-1}\$.
3. For any two suffixes $suf_i$ and $suf_j$ of $x\$, if we let $w_{ij}$ be the common prefix of $suf_i$ and $suf_j$, then the path in $T_x$ relative to $w_{ij}$ is the same for $suf_i$ and $suf_j$.

You should verify that the tree we drew above for $x = abaab$ indeed satisfies these three properties.