10.1 Recognition of Substrings: Recognizing Patterns in Text

We will consider one important special case of pattern matching, involving a set pattern and given text. One string is the \textbf{pattern}, \( p = p_1 \ldots p_l \) and another is a sample of \textbf{text} \( t = t_1 \ldots t_n \). Both the pattern and text are comprised of letters from an \textbf{alphabet} \( A \). Thus \( p_i \in A \) and \( t_j \in A \) for all \( 1 \leq i \leq l \) and \( 1 \leq j \leq n \).

\textbf{Goal:} Construct from pattern \( p \) a deterministic pattern-matching machine \( M_p \) that recognizes the shortest instance of a string in \( A^* \). (Note: \( A^* \) is the set of all strings that can be constructed using the characters in the alphabet \( A \)).

10.2 A First Attempt: “Skeletal DFA”

To construct \( M_p \), we first construct a “skeletal” DFA with \( l + 1 \) states labeled 0, 1, 2, \ldots, \( l \) and a transition from state \( i - 1 \) to state \( i \) on input symbol \( p_i \) (the \( i \)th character in the pattern \( p \)). The skeletal DFA is pictured below:

\[
\begin{array}{c}
\text{start} \\
0 \quad p_1 \quad 1 \quad p_2 \quad 2 \quad \ldots \quad l - 1 \quad p_l \quad l
\end{array}
\]

State 0 has a transition to itself on all symbols \( p_j \neq p_1 \). In general we can think of the \( i \)th state as a pointer to the \( i \)th position in the pattern \( p \).

10.3 Constructing a Pattern Matching Machine

We want the pattern matching machine \( M_p \) to operate like a deterministic finite automaton, \textit{except} that it can make general static transitions while scanning the same input symbol. Do not get too worried about the technicality of this statement. Instead focus on the key aspects that we are hoping to achieve with \( M_p \). We want it to be \textbf{deterministic}, meaning that it will always act the same way and produce the same output when given a specific input. We want it to be \textbf{finite}, meaning that it has a set number of states. It cannot have an infinite number of states, or a variable number of states based on the input. Finally it makes \textbf{state transitions}, which means that the machine keeps track of which of its states it is currently in, and
transitions states every time it consumes a new symbol from the input. States can transition to themselves, so the machine does not always need to change state when a new symbol is consumed.

In terms of structure, $M_p$ has the same set of states as the skeletal machine. Key aspects of $M_p$:

- The state $j$ of $M_p$ corresponds to the prefix $p_1p_2...p_j$ of the pattern $p$.
- $M_p$ starts in State 0 with its “pointer” (what keeps track of its current state) at $t_1$, the first symbol of the input text. Remember the input text is $t = t_1...t_n$.
- If $t_1 = p_1$ (the first symbol of the input is equal to the first symbol of the pattern we are searching for), then $M_p$ remains in State 0 and advances its pointer to position 2 in the text (ie. $t_2$).
- If $t_1 \neq p_1$, then $M_p$ remains in State 0 and advances its input pointer to position 2 in the text (ie. $t_2$).

Now suppose after having read $t_1t_2...t_k$ (the first $k$ characters of the text $t$) we find that $M_p$ is in State $J$. This implies that the last $j$ symbols of $t_1t_2...t_k$ are $p_1p_2...p_j$ and the last $m$ symbols of $t_1t_2...t_k$ are not a prefix of $p = p_1...p_l$ for $m > j$. But what about the case where $t_{k+1} \neq p_{j+1}$? That is, when the next symbol of the text is not the next symbol in the pattern. In this case $M_p$ enters the highest number state $i$ such that $p_1p_2...p_i$ is a suffix of $t_1t_2...t_{k+1}$. To help determine $i$, the machine $M_p$ has associated with it an integer valued function $f$. This function $f$ is called the **failure function** for the pattern $p$. We define $f$ such that $f(j)$ is the largest integer $s$ less than $j$ for which $p_1...p_s$ is a suffix of $p_1p_2...p_j$. That is, $f(j)$ is the largest $s < j$ such that $p_1p_2...p_s = p_{j-s+1}p_{j-s+1}...p_j$. In words, $s$ is the number of positions we can ”fall back” to keep looking for the pattern $p$, based on the fact that the suffixes of some prefixes of $p$ are prefixes of $p$ itself. If there is no such $s \geq 1$, then $f(j) = 0$. The next lecture will cover how to compute the failure function and use it to run the wondrous Knuth-Morris-Pratt algorithm.