A Counting Problem

Last time I described various decision questions concerning regular languages: do two DFAs accept the same language, does a given regular expressions denote Σ^*, and so on. But not everything we'd like to figure out is a decision question. Today I'd like to show an example of a *counting problem* dealing with regular languages.

Here is the problem: given a DFA M and a number n, how many strings of length n are in $L(M)$? In other words, what is the cardinality of $L(M) \cap \Sigma^n$?

One approach is just to generate *all* strings of length n and then test, for each, if it is in $L(M)$. This works, but it takes exponential time—at least 2^n time if $|\Sigma| \geq 2$. Is there a more efficient way?

A more efficient solution uses *dynamic programming*. Don't let the word scare you. All I mean is that we should think recursively.

Here's a recursive decomposition for the problem. Fix a DFA $M = (Q, \Sigma, \delta, q_0, F)$ and the number n. Then let's define

$$f(q, i) = \text{ the number of strings } x \in \Sigma^i \text{ such that } \hat{\delta}(q_0, x) = q$$

for any $q \in Q$ and $i \geq 0$. Here $\hat{\delta}$ is the extension of δ to the domain $Q \times \Sigma^*$, as we defined earlier in our class. So $f(q, i)$ counts the number of ways you get to state q, starting at the start state, by processing strings of length i.

What I'd like you to notice is that we can write a simple recursive expression for f. Namely, as a base case, note that

$$f(q, 0) = \begin{cases} 1 & \text{if } q = q_0 \\ 0 & \text{if } q \neq q_0 \end{cases}$$

since one string of length 0, the empty string, takes you from state q_0 to q_0, while no string of length 0 takes you from q_0 to some *other* state q. In addition, for $i \geq 1$ we have that

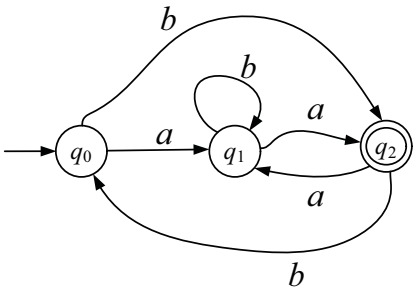
$$f(q, i) = \sum_p f(p, i-1)$$

where the sum is taken over all $p \in Q$ such that $\delta(p, a) = q$ for some a. If there are $j > 1$ characters a such that $\delta(p, a) = q$, then we mean to include all j times in the sum above. Formally, you could say that the sum is actually over $(p, a) \in Q \times \Sigma$ such that $\delta(p, a) = q$. The explanation for the recursive equation above is this: the number of ways to reach q from q_0 by a path of length i if just the number of ways to reach potential predecessor states from q_0 by a path of length $i - 1$, where a "predecessor state" is a state p and a character a such that $\delta(p, a) = q$.

Our algorithm, then is just to compute $f(p, i)$ for all $p \in Q$ and $1 \leq i \leq n$. The answer to the originally asked question will then be

$$\text{answer} = \sum_{q \in F} f(q, n) \ .$$

Let's do a little example, answering how many strings of length $n = 6$ the following DFA M accepts:



Consulting our formula, we can create a little table of $f(q, i)$ values, as follows:

i	$f(q_0, i)$	$f(q_1, i)$	$f(q_2, i)$
0	1	0	0
1	0	1	1
2	1	2	1
3	1	4	3
4	3	8	5
5	5	16	11
6	11	32	21

The first row is filled in according to the base case. After that, for this DFA, the leftmost entry of each row is the rightmost entry of the prior row, while the middle entry of each row is the sum of all three entries in the prior rows, and the rightmost entry of each row is the sum of the prior two entries in the prior row. Note that, as expected (why?), the sum of entries in row i is 2^i. The answer to the originally asked question is 21, the sum of the entries corresponding to final states for row-6. (Note: in class I finalized state q_0, too, so the answer was then $11 + 21 = 32$.)

Let's build on the above to solve a related problem: instead of counting the number of strings of a given length in $L(M)$, let's try to uniformly generate a random string of length n that's in $L(M)$. This could be done as follows—illustrated again for the example above. To uniformly generate a random string of length $n = 6$, uniformly choose a random number x between 1 and 21 (since we know there are 21 strings of length 6 in the language). Suppose, for example, we choose $x = 11$. Then we would like to generate the 11th string of length 6 in the language, with respect to some arbitrary but fixed way of numbering the 21 strings in the language of length 6. Well, where do the 21 strings come from? Consulting our chart and the DFA, some 5 of them end at q_2 with a final transition of b from state q_0, while another 16 of them end at q_2 with a final transition of a from state q_1. We want the 11th string, so we could consider the first five choices to be the b-terminal strings from q_0 and the next 16 choices to be the a terminal strings from q_1. So the final character for us is a and we now want to 6th ($6 = 11 - 5$) string of length 5 that would terminate at q_1. Continue unwinding in this way to recover the entire string. At this point we have shown how to solve the random-generation problem.

Let's look at one final counting problem: count the number of strings of length n in $L(\alpha)$ for some regular expression α. We could convert α to a DFA and use the procedure we have developed already, but the conversion will run in exponential time. Is there a more efficient approach? The answer seems to be *no*. Nobody has developed such a procedure, and most people would find it surprising if such a procedure was found to exist.