18 More String Matching (April 10)

18.1 Redundant Comparisons

Let's go back to the character-by-character method for string matching. Suppose we are looking for the pattern 'ABRACADABRA' in some longer text using the (almost) brute force algorithm described in the previous lecture. Suppose also that when $s = 11$, the substring comparison fails at the fifth position; the corresponding character in the text (just after the vertical line below) is not a C. At this point, our algorithm would increment s and start the substring comparison from scratch.

HOCUSPOCUSABRA|BRACADABRA...
ABRA|~~CADABRA~~
ABR|ACADABRA

If we look carefully at the text and the pattern, however, we should notice right away that there's no point in looking at $s = 12$. We already know that the next character is a B — after all, it matched $P[2]$ during the previous comparison — so why bother even looking there? Likewise, we already know that the next two shifts $s = 13$ and $s = 14$ will also fail, so why bother looking there?

HOCUSPOCUSABRA|BRACADABRA...
ABRA|~~CADABRA~~
~~ABR|ACADABRA~~
~~AB|RACADABRA~~
A|BRACADABRA

Finally, when we get to $s = 15$, we can't immediately rule out a match based on earlier comparisons. However, for precisely the same reason, we shouldn't start the substring comparison over from scratch — we already know that $T[15] = P[4] =$ A. Instead, we should start the substring comparison at the *second* character of the pattern, since we don't yet know whether or not it matches the corresponding text character.

If you play with this idea long enough, you'll notice that the character comparisons should always advance through the text. **Once we've found a match for a text character, we never need to do another comparison with that character again.** In other words, we should be able to optimize the brute-force algorithm so that it always *advances* through the text.

You'll also eventually notice a good rule for finding the next 'reasonable' shift s. A *prefix* of a string is a substring that includes the first character; a *suffix* is a substring that includes the last character. A prefix or suffix is *proper* if it is not the entire string. Suppose we have just discovered that $T[i] \neq P[j]$. **The next reasonable shift is the smallest value of s such that $T[s .. i-1]$, which is a suffix of the previously-read text, is also a proper prefix of the pattern.**

In this lecture, we'll describe a string matching algorithm, published by Donald Knuth, James Morris, and Vaughn Pratt in 1977, that implements both of these ideas.

18.2 Finite State Machines

If we have a string matching algorithm that follows our first observation (that we always advance through the text), we can interpret it as feeding the text through a special type of *finite-state*