String-search Alternatives to the KMP Algorithm Tyler Neylon

This is a collection of thoughts on ways to solve the following problem: given a long string t (the *text*), and a shorter string s, find the first occurrence of s in t. This problem can be solved in $O(T)$ time, where $T = |t|$, using the Knuth-Morris-Pratt, or KMP, algorithm. We need at least T time to simply read the text t, so this can be seen as an optimally-fast deterministic algorithm. I am interested in ways to improve upon some features of the KMP algorithm, however. For now, there is only one idea in this paper, but I'll leave open the possibility of adding more later.

I'm not doing background research to check the originality of these ideas. These are independent thoughts. If it turns out they have been published by others previously, I can add the details to this paper when I find out about the previous work.

1. Incrementally-computed hash table

Let S denote the length of s. In this algorithm, we create a hash table from all S-length substrings of t to the index of that substring within t. Then looking for s in t is simply a hash table lookup.

This can be fast if we build such a hash table in time $O(T)$, which is what we'll do. The naive way to build the table would require time $O(ST)$, as we would normally need time $O(S)$ to compute the hash of each length-S substring of t.

We'll choose positive integer parameters x and N, and use the following hash function, illustrated with $s' = s_0 s_1 s_2 s_3$:

$$h(s') = s_0 + s_1 x + s_2 x^2 + s_3 x^3. \pmod N$$

So h is a polynomial with coefficients given by the characters of s'.

Now consider any adjacent substrings s' and s'' in t. If we write the i^{th} character of t as t_i, so that

$$t = t_0\ t_1\ \cdots\ t_{T-1},$$

then an example of adjacent s' and s'' would be given by

$$s' = t_3\ t_4\ t_5\ t_6$$

and

$$s'' = t_4\ t_5\ t_6\ t_7.$$

We then have

$$h(s') = t_3 + t_4 x + t_5 x^2 + t_6 x^3 \pmod N$$

and

$$h(s'') = t_4 + t_5 x + t_6 x^2 + t_7 x^3. \pmod N$$

So that

$$h(s') = \big(h(s'') - t_7 x^3\big)x + t_3. \pmod N$$

In other words, knowing $h(s'')$ and the position of s'' in t allows us to compute $h(s')$ in constant time.

We can find the hash of the last length-S substring in t, and build the hash table by working backwards from there, one adjacent substring at a time. This gives us the desired hash table in $O(T)$ time, and is the complete algorithm.

This algorithm is technically probabilistic. In practice, hash tables are considered to be reliably fast. I see the advantage of this algorithm as its conceptual simplicity. I consider it easier to learn and remember than the KMP algorithm.

To further study this algorithm, it would be good to verify that the proposed hash function uniformly distributes its outputs, so that unwanted hash collisions are effectively avoided.

2. Edit this paper

The LaTeX source for this paper is hosted at http://github.com/tylerneylon/math/, and may be modified and redistributed freely, as long as the original source is credited. The author will happily accept, with gratitude, git pull requests that improve the paper.