Algorithms in Systems Engineering
ISE 172

Lecture 24

Dr. Ted Ralphs

References for Today's Lecture

- Required reading
  - Section 8.6
- References
  - CLRS Chapter 32
  - R. Sedgewick, *Algorithms in C++* (Third Edition), 1998.

The Knuth-Morris-Pratt Algorithm

- By being a little smarter, we can get a string-matching algorithm linear run time.
- The Knuth-Morris-Pratt Algorithm is based on the pre-computation a *prefix function* for the pattern.
- The prefix function "looks ahead" and tells us something about the validity of upcoming shifts by examining how well the current shift matches the pattern.
- Example:

The Naive KMP Algorithm

- In this simplified algorithm, we keep track of two pointers s and r.
- The pointer s is the start of the shift we are investigating.
- At all times, we maintain r such that $P[0..r-s] = T[s..r]$.
- The Simple KMP Algorithm

  1. Set $s = 0$ and $r = -1$.
  2. While $s < n - m$
     - If $r - s = m - 1$, then s is a valid shift. Increase s by at least one and continue.
     - If $P[r - s + 1] = T[r + 1]$, then increase r by 1 and continue.
     - If $P[r - s + 1] \neq T[r + 1]$, then increase s until $P[0..r - s] = T[s..r]$ and continue.

- How do we implement the third option in Step 2?
- What is the running time of this algorithm?

Improving the Simple KMP Algorithm

- Suppose that the string $P[0..q]$ matches $T[s..s+q]$, but $P[q+1] \neq T[s+q+1]$.
- Question: What is the smallest k such that $P[0..q-k] = T[s+k, s+q]$?
- In other words, what is the first shift after s that isn't necessarily invalid based on our knowledge of $T[s..s+q]$?
- The answer has to do with how the pattern overlaps itself.
- We can easily determine the answer for a particular instance, but how do we do so efficiently?

The Prefix Function

- A *prefix* of a string S is any substring beginning with the first character of S, i.e., $S[0..q]$.
- A *suffix* is defined similarly as any substring of S that ends with the last character of S.
- Note that the empty string is a both prefix and a suffix for any string.
- Define $\pi[q]$ to be the length of the longest prefix of P that is a suffix of $P[0..q]$.
- The answer to the question from the previous slide is then $\rho(q) = q - \pi[q] + 1$ (why?).
- Note that the value of $\rho[q]$ depends only on P, not on T.
- We can calculate $\rho[0..m-1]$ easily in $\Theta(m)$ time.
- What do we do with this?

Example: The Prefix Function

i	-1	0	1	2	3	4	5	6	7	8	9	10	11	12
$P[i]$		P	A	R	T	I	C	I	P	A	T	I	O	N
$\pi[i]$	-1	0	0	0	0	0	0	0	1	2	0	0	0	0
$\rho[i]$	1	1	2	3	4	5	6	7	7	7	10	11	12	13

i	13	14	15	16	17	18	19	20	21	22	23	24	25
$P[i]$		I	S		P	A	R	A	M	O	U	N	T
$\pi[i]$	0	0	0	0	1	2	3	0	0	0	0	0	0
$\rho[i]$	14	15	16	17	17	17	20	21	22	23	24	25	26

Computing The Prefix Function

- Although the length of the search pattern is often small, we still want to compute the prefix function as efficiently as possible,
- In particular, we would like to be able to compute it in linear time if we can.
- Note the pattern in the example: we always have $\pi[i] \leq \pi i - 1 + 1$.
- Just as the length of the match of the search pattern against the string can only increase by one in each iteration, we have a similar phenomena with the length of the longest matching suffix of $P[0..q]$.
- For each value of q, we need only check whether the longest matching suffix for position $q - 1$ can be extended to position q.
- Thus, $\pi[q]$ is either $\pi[q - 1] + 1$ or 0.

The Algorithm

- In the KMP algorithm, we keep track of two pointers s and r.
- The pointer s is the start of the shift we are investigating.
- At all times, we maintain r such that $P[0..r-s] = T[s..r]$.
- For notational convenience, set $\rho(-1) = 1$.
- The KMP Algorithm

  1. Set $s = 0$ and $r = -1$.
  2. While $s < n - m$
     - If $r - s = m - 1$, then s is a valid shift. Increase s by $\rho(r-s)$ and continue.
     - If $P[r-s+1] = T[r+1]$, then increase r by 1 and continue.
     - If $P[r-s+1] \neq T[r+1]$, then increase s by $\rho(r-s)$ and continue.

- Now what is the running time of the algorithm?