

Building the Finite Automaton (continued from last class) and KMP algorithm

Advanced Algorithms (CSE 794)

Instructor: Tamal K. Dey

Scribe: Wenjing Ma

May 6, 2009

1 Building the Finite-Automaton

This section follows the discussion in last class. Now we will introduce how to build the finite automaton. The basic idea is finding the largest length of the suffix of $P_q a$, which is a prefix of P . That is to say, $\delta(q, a) = \sigma(P_q a)$. The algorithm is shown below. For each prefix of P with length q , we find $\delta(q, a)$ for every a in Σ . Line 4 to 7 is the loop that finds $\sigma(P_q a)$. k is set as the length of $P_q a$, then it is decreased until $P_k \sqsupseteq P_q a$.

```
COMPUTE_TRANSITION_FUNCTION( $P, \Sigma$ )
```

1. $m \leftarrow |P|$
2. for $q := 0$ to m
3. for each $a \in \Sigma$
4. $k := \min(m + 1, q + 2)$
5. repeat $k := k - 1$
6. until $P_k \sqsupseteq P_q a$
7. $\delta(q, a) := k$
8. endfor
9. endfor
10. return δ

The loop at Line 5 to Line 6 takes time $O(m)$, and the comparison for $P_k \sqsupseteq P_q a$ also takes time $O(m)$. The loop from Line 3 to Line 8 takes time $O(|\Sigma|)$, and the “for” loop from Line 2 to Line 9 has an order of m . Thus, the total time of constructing the transition function is $O(m^3 |\Sigma|)$.

2 KMP Algorithm

KMP algorithm can be described in this way. Suppose

$P[1, \dots, q] = T[s + 1, \dots, s + q]$,
 $P[1, \dots, k] = T[s' + 1, \dots, s' + k]$, where $s' + k = s + q$. The aim is looking for largest $k < q$, such that $P_k \sqsupset P_q$.

2.1 Prefix function

The improvement of KMP over Finite Automaton algorithm is that it removes some useless states from the transition function. In the preprocessing, there are only m entries, instead of $m \cdot |\Sigma|$. For position i in P , the $\pi(i)$ value determines the largest prefix of P_{i-1} that is a suffix of P_i . Thus, the prefix function is calculated in the following way.

$$\pi : 1, 2, \dots, m \rightarrow 0, 1, \dots, m - 1.$$

$$\pi(q) = \max\{k \mid k < q \text{ and } P_k \sqsupset P_q\}.$$

The algorithm is shown below. In the while loop of Line 5 to Line 7, while $P[k + 1] \neq P[q]$ and $k > 0$, k is set to $\pi(k)$. If $P[k + 1] = P[q]$, it means we can enlarge the matched prefix of P by including $P[k + 1]$, so $k := k + 1$ (Line 8). Then $\pi(q) := k$ (Line 10).

```

COMPUTE_PREFIX_FUNCTION(P)
1. m ← | P |
2. π[1] ← 0
3. k ← 0
4. for q:= 2 to m
5.   while k > 0 and P[k + 1] ≠ P[q]
6.     k := π(k)
7.   endwhile
8.   if P[k + 1] = P[q] then k:=k+1
9.   endif
10.  π(q) := k
11. endfor
12. return π
  
```

To show how it works, let us look at Figure 1. In this example, the pattern P is “ababababca”, whose length is 10. So, consider q with value 1 to 10, and we will have the following results.

For $q = 1$, $\pi(1) = 0$;

For $q = 2$, we consider “a” as P_1 , and “ab” as P_2 . Since P_1 is not a suffix of P_2 , $\pi(2)$ is 0;

For $q = 3$, we consider “a” as P_1 , “ab” as P_2 , and “aba” as P_3 . It can be seen that P_1 is a suffix of P_3 , but P_2 is not. So, 1 is the maximum value of k , for which P_k is a suffix of P_3 . Thus, $\pi(3) = 1$.

...

After calculating all the 10 values for q , we came up with the table in Figure 2.

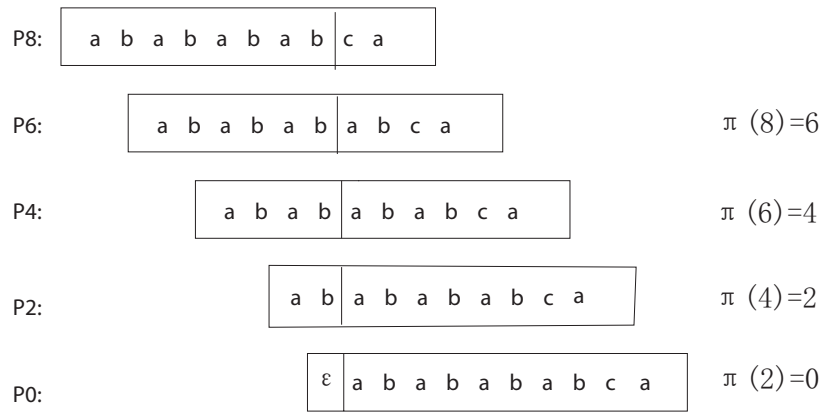


Figure 1: The process of computing π function for “ababababca”

i	1	2	3	4	5	6	7	8	9	10
$P(i)$	a	b	a	b	a	b	a	b	c	a
$\pi(i)$	0	0	1	2	3	4	5	6	0	1

Figure 2: The values of π function for “ababababca”

2.2 KMP matching function

Suppose we have already computed the prefix function. Then, the matching function will scan the string in a way similar to the COMPUTE_PREFIX_FUNCTION.

The following algorithm shows the matching function. The difference from COMPUTE_PREFIX_FUNCTION is that instead of checking each element in P , we are checking T now. So in KMP_MATCHER, on Line 6 and 9, we use $T[i]$ instead of the $P[q]$ used in COMPUTE_PREFIX_FUNCTION. If $q = m$, it means we have found a matching.

KMP_MATCHER(T, P)

1. $n := |T|$
2. $m := |P|$
3. $\pi = \text{COMPUTE_PREFIX_FUNCTION}(P)$
4. $q := 0$
5. for $i := 1$ to n
6. while $q > 0$ and $P[q + 1] \neq T[i]$
7. $q := \pi(q)$
8. endwhile
9. if $P[q + 1] = T[i]$ then $q := q + 1$
10. endif
11. if($q = m$) then
12. print “match with shift $i - m$ ”
13. $q := \pi(q)$
14. endif
15. endfor

There are n iterations in the “for” loop from line 5 to 14 . To compute the running time of each iteration, according to the amortized analysis, we can choose the value of q as the potential. The decrease of q in line 6 to 8 will never exceed the increase of q in Line 9 to Line 10, which increases q only by 1 each time. Since Line 9 to Line 10 costs $O(1)$ time, the “while” loop in line 6 to 8 also has amortized running time of $O(1)$. Thus, the time of the whole “for” loop is $O(n)$. For the same reason, the time of the $\text{COMPUTE_PREFIX_FUNCTION}(P)$ has running time $O(m)$.

References

- [1] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms. *Second Edition*, The MIT Press(2001), 922–927.