String Matching: KMP Algorithm

π table = Longest prefix = Longest suffix

a b c d a b c.

Prefix: a, ab, $\boxed{abc}$, abcd, ---

Suffix: c, bc, $\boxed{abc}$, dabc, ---

abc is common.

(1)

1	2	3	4	5	6	7	8	9	10
a	b	c	d	a	b	e	a	b	f
0	0	0	0	1	2	0	1	2	0

(2)

1	2	3	4	5	6	7	8	9	10	11
a	b	c	d	e	a	b	f	a	b	c
0	0	0	0	0	1	2	0	1	2	3

(3)

1	2	3	4	5	6	7	8	9
a	a	a	a	b	a	a	c	d
0	1	2	3	0	1	2	0	0

4

	1	2	3	4	5	6	7	8	9	10
	a	b	a	b	a	b	a	b	c	a
	0	0	1	2	3	4	5	6	0	1

The *next* function

Given $P[1..m]$, let *next* be a function $\{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$ such that

next(q)$= max\{k : k < q$ and $P[1..k]$ is a suffix of $P[1..q]\}$.

q	1	2	3	4	5	6	7	8	9	10
P[q]	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*c*	*a*
next(q)	0	0	1	2	3	4	5	6	0	1

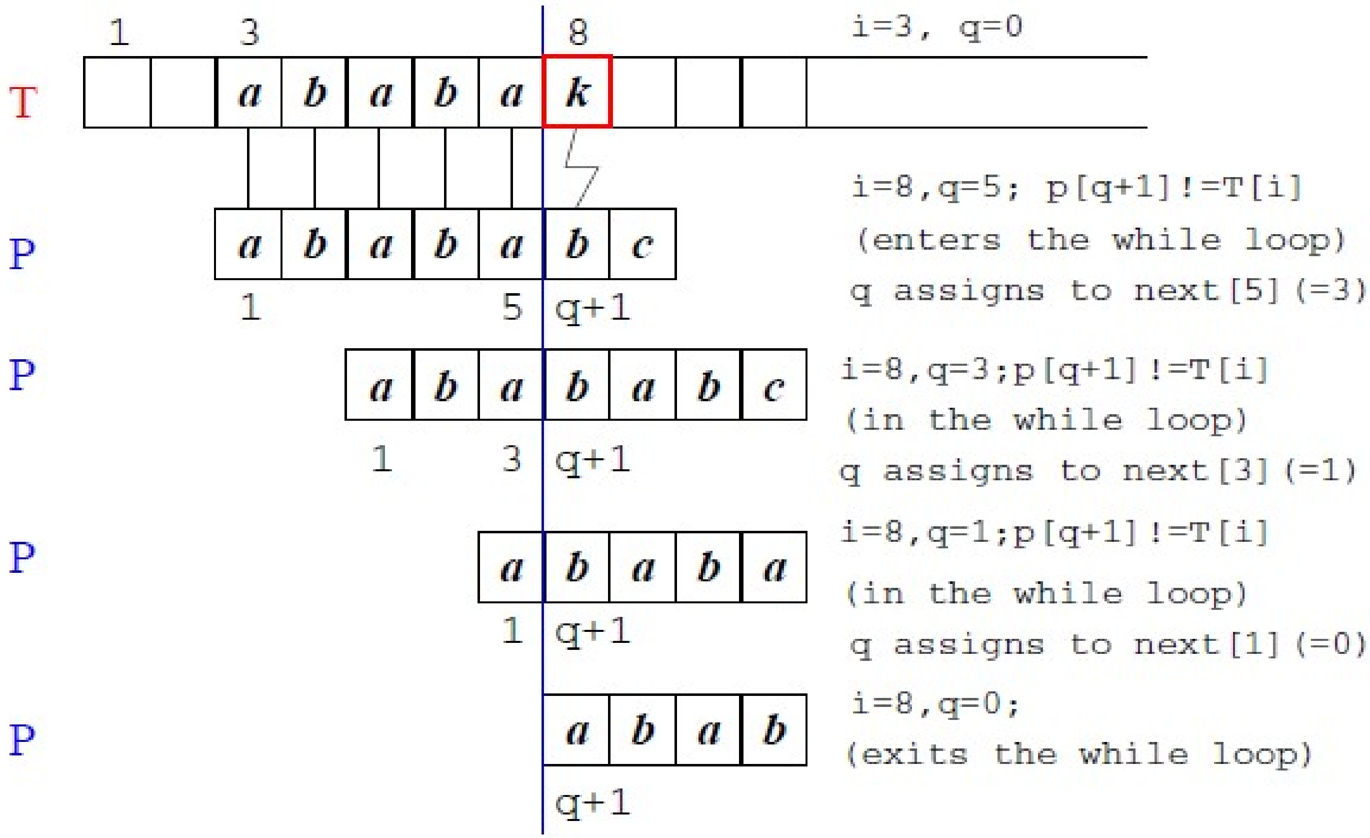
Given *next(q)* for all $1 \leq q \leq m$, we can use the KMP algorithm.

The Knuth-Morris-Pratt (KMP) Algorithm

```
KMP_StringMatcher(T, P) {
 n = length(T); m = length(P);
 compute_Next(P);
 q = 0; // number of characters matched
        // so far
 i=1;
 while (i<=n) {
  // loop until a match is found, or
  // number of characters matched so far
  // is 0; note 'i' is unchanged.
  while (q > 0 and P[q+1] != T[i]) {
   q=next[q];
  }
  // matched character increased by 1
  if (P[q+1]==T[i]) q=q+1;
  if (q==m) {
   print "Pattern occurs with shift=", i-m
   q=next[q];
  }
  i++;
 }
}
```

The Knuth-Morris-Pratt (KMP) Algorithm

q	1	2	3	4	5	6	7
P[q]	*a*	*b*	*a*	*b*	*a*	*b*	*c*
next(q)	0	0	1	2	3	4	0



How to compute *next* function

We first set `next[1]=0`, then compute $next[q]$ with $q = 2, 3, \ldots m$, one by one in $m - 1$ iterations.

```
compute_Next(P) {
 m = length(P);
 next[1]=0; // initialization
 k = 0; // number of characters matched
        // so far
 q=2;
 while (q<=m) {
  while (k > 0 and P[k+1] != P[q]) {
   k = next[k];
  }
  if (P[k+1]==P[q]) k=k+1;
  next[q]=k;

  q++;
 }
```

KMP Algorithm

i

T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

j

	0	1	2	3	4	5
P		a	b	a	b	d
π		0	0	1	2	0