Knuth-Morris-Pratt (KMP) matcher

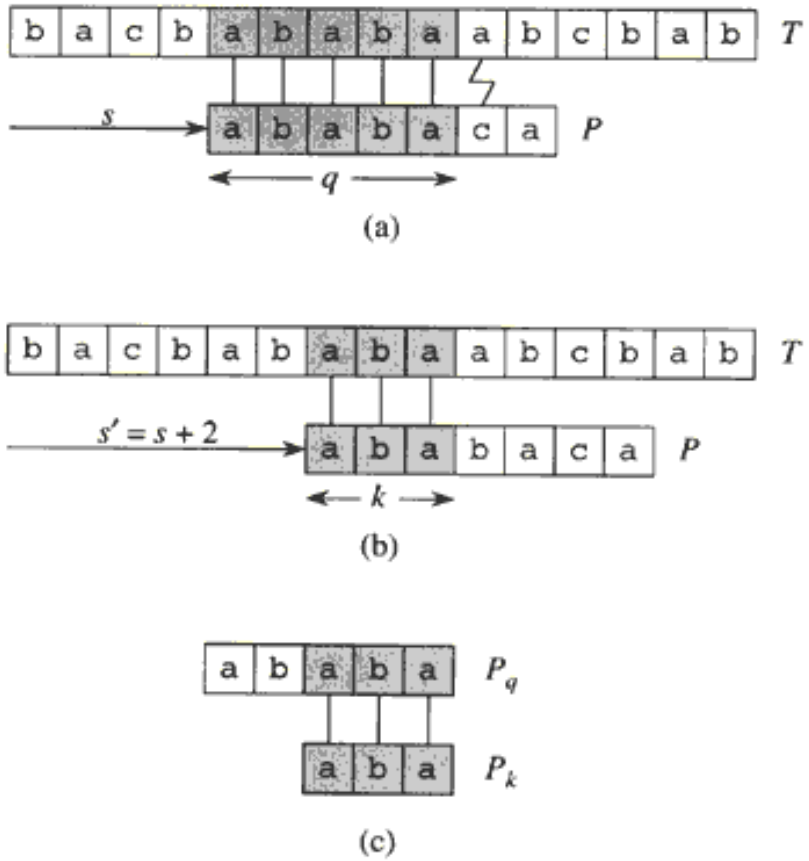
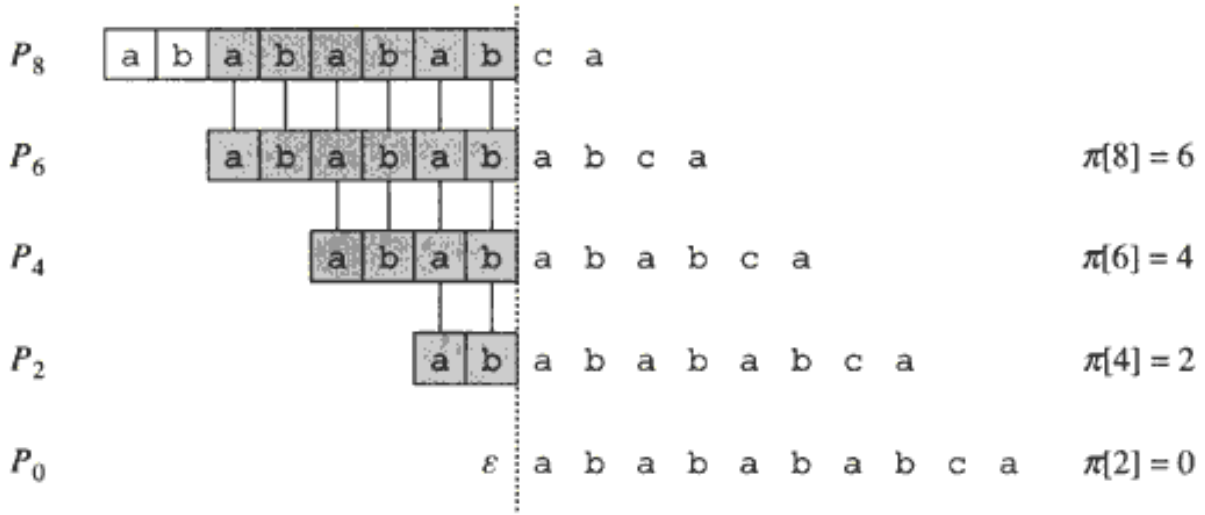


Figure 32.10 The prefix function π. **(a)** The pattern $P = \texttt{ababaca}$ is aligned with a text T so that the first $q = 5$ characters match. Matching characters, shown shaded, are connected by vertical lines. **(b)** Using only our knowledge of the 5 matched characters, we can deduce that a shift of $s + 1$ is invalid, but that a shift of $s' = s + 2$ is consistent with everything we know about the text and therefore is potentially valid. **(c)** The useful information for such deductions can be precomputed by comparing the pattern with itself. Here, we see that the longest prefix of P that is also a proper suffix of P_5 is P_3. This information is precomputed and represented in the array π, so that $\pi[5] = 3$. Given that q characters have matched successfully at shift s, the next potentially valid shift is at $s' = s + (q - \pi[q])$.

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

(a)



(b)

Figure 32.11 An illustration of Lemma 32.5 for the pattern $P = \texttt{abababaca}$ and $q = 8$. **(a)** The π function for the given pattern. Since $\pi[8] = 6$, $\pi[6] = 4$, $\pi[4] = 2$, and $\pi[2] = 0$, by iterating π we obtain $\pi^*[8] = \{6, 4, 2, 0\}$. **(b)** We slide the template containing the pattern P to the right and note when some prefix P_k of P matches up with some proper suffix of P_8; this happens for $k = 6$, 4, 2, and 0. In the figure, the first row gives P, and the dotted vertical line is drawn just after P_8. Successive rows show all the shifts of P that cause some prefix P_k of P to match some suffix of P_8. Successfully matched characters are shown shaded. Vertical lines connect aligned matching characters. Thus, $\{k : k < q \text{ and } P_k \sqsupset P_q\} = \{6, 4, 2, 0\}$. The lemma claims that $\pi^*[q] = \{k : k < q \text{ and } P_k \sqsupset P_q\}$ for all q.

Exercice :

Compute the prefix function for the pattern
ababbabbabbababbabb

Compute prefix table bestp

```
bestp[1] = 0
k = 0

for q in range 2 to |P| + 1 :
        while k > 0 and P[k+1] != P[q]:
                k = bestp[k] //next char does not
match, start from a shorter prefix

        if P[k+1] = P[q]:   //next char matches
                k = k + 1
        bestp[q] = k

return bestp
```

KMP - matcher

```
bestp = compute prefix table(P)
q = 0

for i in range 1 to |T| + 1 :
        while q > 0 and P[q+1] != T[i]:
                q = bestp[q] //next char does not
match, start from a shorter prefix

        if P[q+1] = T[i]:
                q = q + 1   //next char matches

        if q = m            //occurrence
                print « occurrence at i – m »
                //start from a shorter prefix to
look for the next match
                q = bestp[q]
```

KMP – matcher analysis

- with an amortized analysis we prove that the compute prefix function time is $O(|P|)$
- in the same manner it can be shown that a KMP search is done in $O(|T|)$ time

Exercice :

Take the pattern ATAGTGCAT and the text given in the pattern_search_dataset file available on moodle.

The goal is to compute a list of positions in the text (0-based index) where the pattern appears.

- implement the naive matcher. Count and print the number of character comparisons
- implement the KMP matcher. Count and print the number of character comparisons for the matching procedure
- compare the two matchers