String Matching

KMP String Matching ,

Design and Analysis of Algorithm
Fall 2022

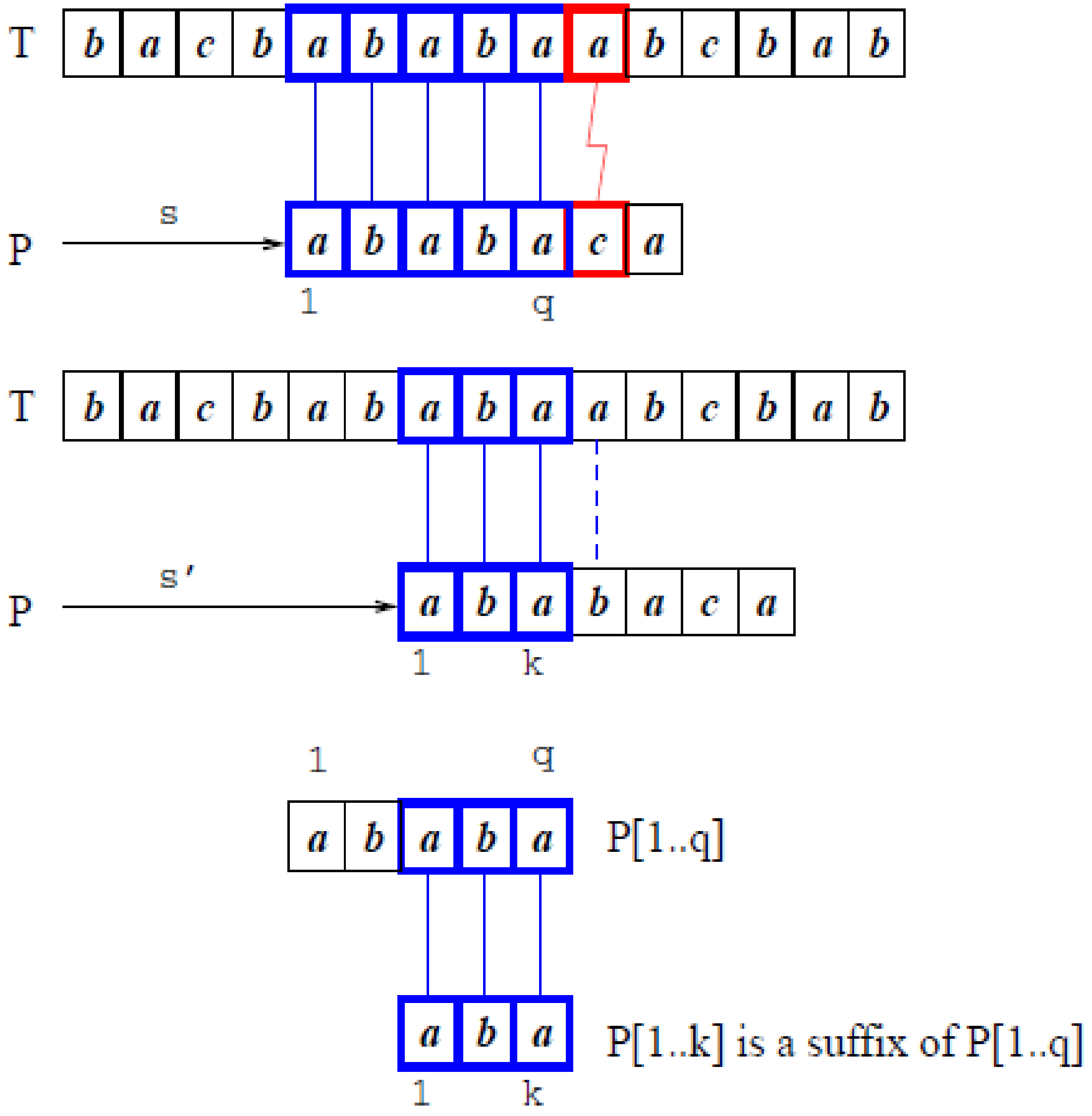
Algorithm	Preprocessing time	Matching time
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\|\Sigma\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$

KMP

Knuth-Morris-Pratt Algorithm

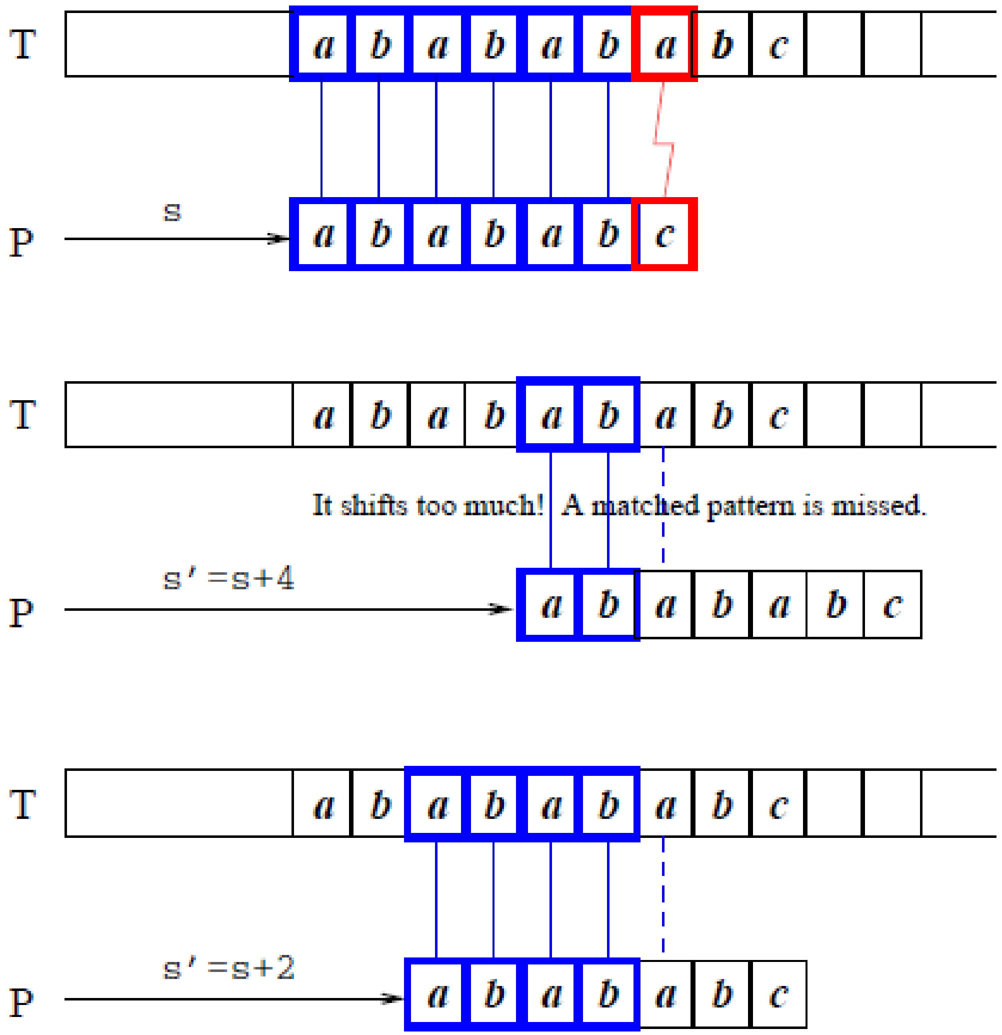
- Searches for occurrences of a pattern x within a main text string y by employing the simple observation: **after a mismatch, the word itself allows us to determine where to begin the next match to bypass re-examination of previously matched characters**
- **Preprocessing phase:** O(m) space and time complexity
- **Searching phase**: Ɵ(n) time complexity (independent from the alphabet size)
- Runs in linear time.
- The algorithm was invented in 1977 by Knuth and Pratt and independently by Morris, but the three published it jointly

When we slide P to right, it should be a place where P could possibly occur in T .



T
b a c b a b a b a a b c b a b
P
s
a b a b a c a
1
q
T
b a c b a b a b a a b c b a b
P
s'
a b a b a c a
1
k
1
q
a b a b a
P[1..q]
a b a
P[1..k] is a suffix of P[1..q]
1
k

Do not shift too much, as it may miss some matched patterns!



Prefix Function Π

- The prefix function π for a pattern holds knowledge about how the pattern matches against shifts of itself.

- This info. can be used to avoid testing useless shifts that the naïve algorithm does

- π contains only m entries, where as δ is a table of $m\,|\sum| = md$ entries

We need to answer the following question: Given $P[1..q]$ match text characters $T[s+1..s+q]$, what is the *least* shift $s' > s$ such that

$P[1..k]$ = $T[s'+1..s'+k]$,

where $s'+k = s+q$?

In practice, the shift s' can be precomputed by comparing P against itself. Observe that $T[s'+1..s'+k]$ is a known text, and it is a **suffix** of $P[1..q]$. To find the *least shift* $s' > s$, it is the same as finding the *largest* $k < q$, s.t.,

$P[1..k]$ is a suffix of $P[1..q]$.

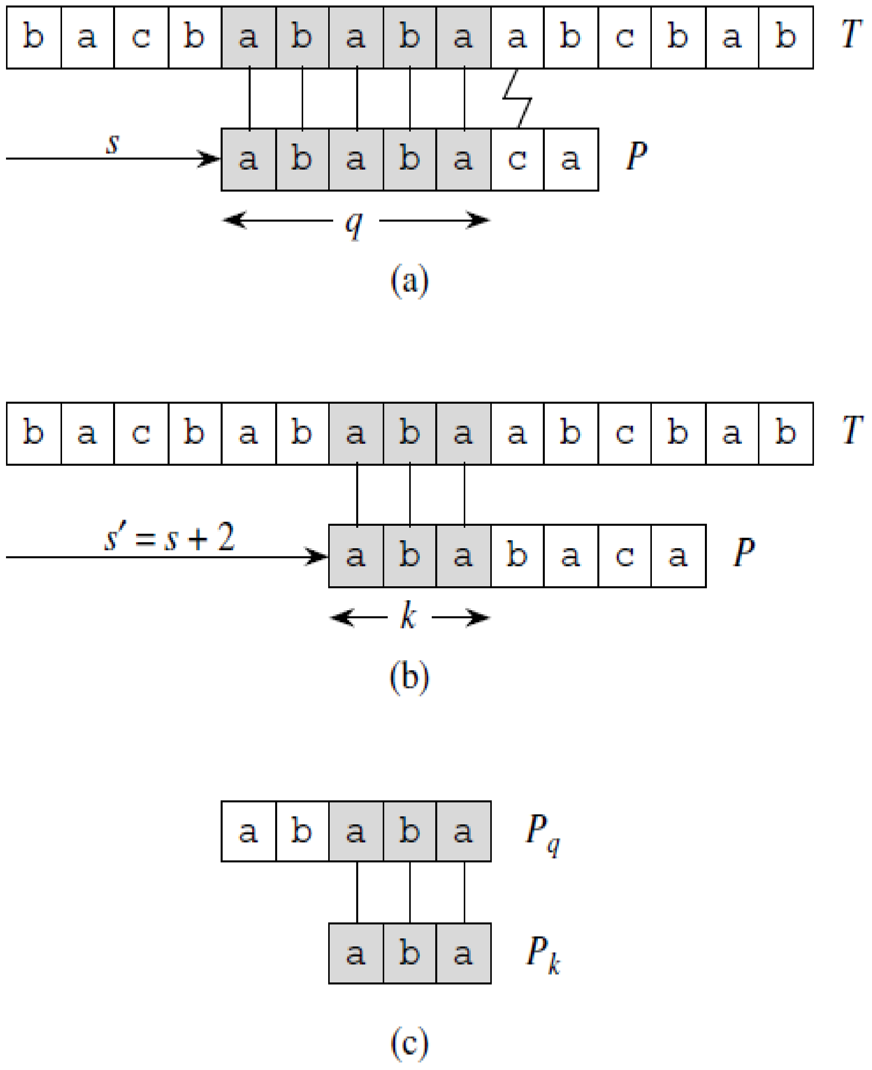


Fig 32.10

If we precompute prefix function of P (against itself), then whenever a mismatch occurs, the prefix function can determine which shift(s) are invalid and directly ruled out. So move directly to the shift which is potentially valid. However, there is no need to compare these characters again since they are equal.

PREFIX FUNCTION

- Figure 32.10 The prefix function π. (a) The pattern P = 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 $P5$ is $P3$. 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 $s = s + (q - \pi[q])$.

Prefix Function Π

- If $P[1..q] = T[s+1..s+q]$, what is the least shift $s' > s$ such that:

$P[1..k] = T[s'+1..s'+k]$ where $s'+k=s+q$?

- Such a shift s' is not necessarily invalid, due to the knowledge of $T[s+1..s+q]$
- In the best case, s' = s+q, ruling out s+1, s+2 ,…,s+q-1.

Prefix Function Π

- This info. can be precomputed by comparing the pattern with itself.

- Given P[1..m], $\pi : \{1,2,\ldots,m\} \rightarrow \{0,1,\ldots m\text{-}1\}$ such that :
- $\pi[q] = max\{\, k : k<q \text{ and } P_k \sqsupset P_q\}$

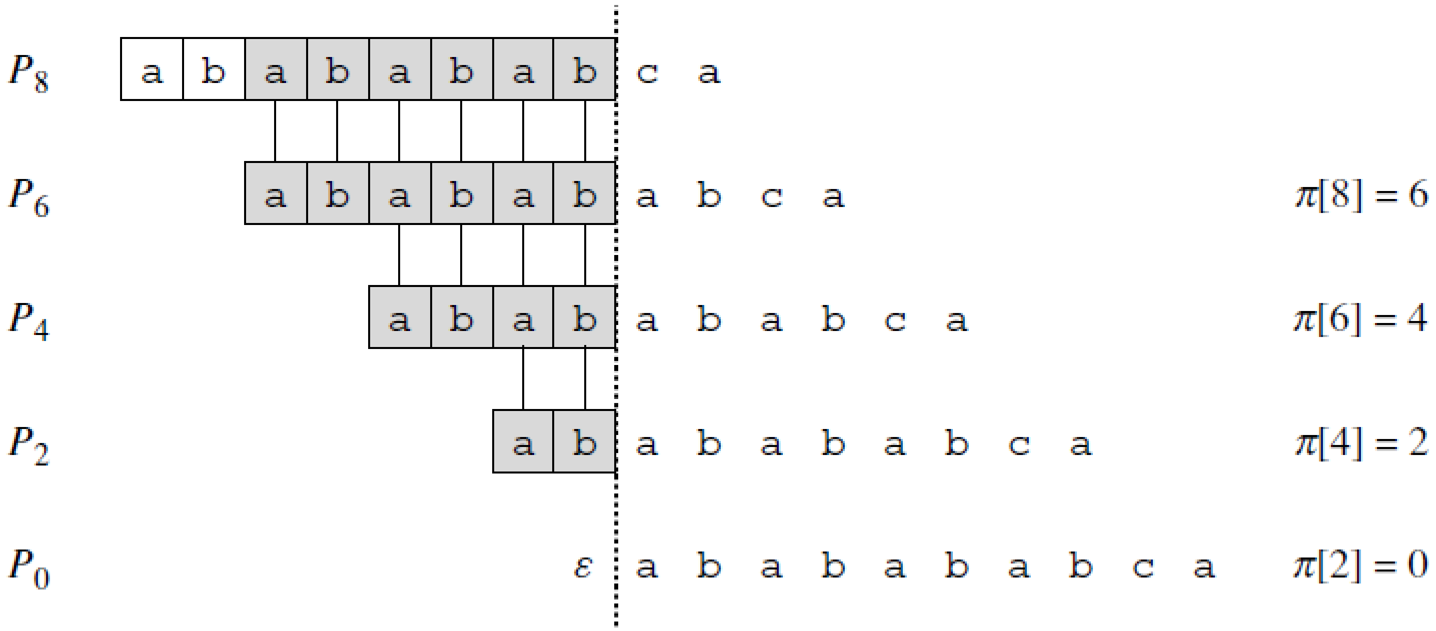
PREFIX FUNCTION Π

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

We slide the template containing the pattern P to the right and note when some prefix Pk of P matches up with some proper suffix of $P8$; this happens for k = 6, 4, 2, and 0

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)



P_8 a b a b a b a b c a
P_6 a b a b a b a b c a $\pi[8] = 6$
P_4 a b a b a b a b c a $\pi[6] = 4$
P_2 a b a b a b a b c a $\pi[4] = 2$
P_0 ε a b a b a b a b c a $\pi[2] = 0$

(b)

Fig 32.11

- Figure 32.11 An illustration of Lemma 32.5 for the pattern P = ababababca 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 Pk of P matches up with some proper suffix of $P8$; 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 $P8$. Successive rows show all the shifts of P that cause some prefix Pk of P to match some suffix of $P8$. Successfully matched characters are shown shaded. Vertical lines connect aligned matching characters. Thus, $\{k : k < q \text{ and } Pk \sqsupset Pq\} = \{6, 4, 2, 0\}$. The lemma claims that $\pi*[q] = \{k : k < q \text{ and } Pk \sqsupset Pq\}$ for all q.

```
KMP-MATCHER(T, P)
1  n = T.length
2  m = P.length
3  π = COMPUTE-PREFIX-FUNCTION(P)
4  q = 0                                       // number of characters matched
5  for i = 1 to n                              // scan the text from left to right
6      while q > 0 and P[q + 1] ≠ T[i]
7          q = π[q]                            // next character does not match
8      if P[q + 1] == T[i]
9          q = q + 1                           // next character matches
10     if q == m                               // is all of P matched?
11         print "Pattern occurs with shift" i − m
12         q = π[q]                            // look for the next match
```

```
KMP-MATCHER(T, P)
1  $n = T.length$
2  $m = P.length$
3  $\pi$ = COMPUTE-PREFIX-FUNCTION(P)
4  $q = 0$                                // number of characters matched
5  for $i = 1$ to $n$                     // scan the text from left to right
6      while $q > 0$ and $P[q+1] \neq T[i]$
7          $q = \pi[q]$                   // next character does not match
8      if $P[q+1] == T[i]$
9          $q = q + 1$                    // next character matches
10     if $q == m$                        // is all of $P$ matched?
11         print "Pattern occurs with shift" $i - m$
12         $q = \pi[q]$                   // look for the next match
```

```
COMPUTE-PREFIX-FUNCTION(P)
  $m \leftarrow length[P]$
  $\pi[1] \leftarrow 0$
  $k \leftarrow 0$
  for $q \leftarrow 2$ to $m$
      do while $k > 0$ and $P[k+1] \neq P[q]$
             do $k \leftarrow \pi[k]$
         if $P[k+1] = P[q]$
           then $k \leftarrow k + 1$
         $\pi[q] \leftarrow k$
  return $\pi$
```

The running time of COMPUTE-PREFIX-FUNCTION is $\Theta(m)$.

The running time of COMPUTE-PREFIX-FUNCTION is $\Theta(m)$, which we show by using the aggregate method of amortized analysis (see Section 17.1). The only tricky part is showing that the **while** loop of lines 6–7 executes $O(m)$ times altogether. We shall show that it makes at most $m - 1$ iterations. We start by making some observations about k. First, line 4 starts k at 0, and the only way that k increases is by the increment operation in line 9, which executes at most once per iteration of the **for** loop of lines 5–10. Thus, the total increase in k is at most $m - 1$. Second, since $k < q$ upon entering the **for** loop and each iteration of the loop increments q, we always have $k < q$. Therefore, the assignments in lines 3 and 10 ensure that $\pi[q] < q$ for all $q = 1, 2, \ldots, m$, which means that each iteration of the **while** loop decreases k. Third, k never becomes negative. Putting these facts together, we see that the total decrease in k from the **while** loop is bounded from above by the total increase in k over all iterations of the **for** loop, which is $m - 1$. Thus, the **while** loop iterates at most $m - 1$ times in all, and COMPUTE-PREFIX-FUNCTION runs in time $\Theta(m)$.

Exercises

32.4-1

Compute the prefix function π for the pattern

ababbabbabbababbabb.

Visualizaers

- https://people.ok.ubc.ca/ylucet/DS/KnuthMorrisPratt.html
- http://whocouldthat.be/visualizing-string-matching/
- http://jovilab.sinaapp.com/visualization/algorithms/strings/kmp

Reference

- **Introduction to Algorithms**
  - Thomas H. Cormen
  - Chapter # 32