Analysis of Algorithms

String Matching

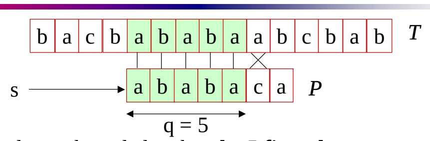
The Knuth-Morris-Pratt Algorithm

- String matching algorithm that runs in linear time ($O(n+m)$) by avoiding the computation of the transition function δ
  - Pattern matching is done using an auxiliary prefix function $\pi[1..m]$ precomputed from the pattern in time $O(m)$.
  - The array $\pi[1..m]$ allows an efficient computation "on-the-fly" of the transition function δ.

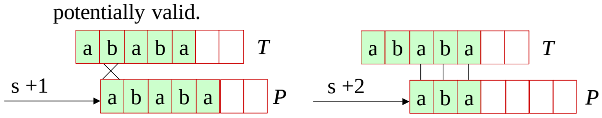
The Knuth-Morris-Pratt Algorithm

- For any state $q = 0, 1, \ldots, m$, and any character $c \in \Sigma$, the value $\pi[q]$ contains the information needed to compute $\delta(q, c)$ that is independent of c.
- Prefix function p[q] $\Rightarrow O(m)$ (substantial savings, particularly if $|\Sigma|$ is large)
- Transition function $\delta[q, c] \Rightarrow O(m|\Sigma|)$

Key Idea



- Using only our knowledge that **the 5 first characters matched**, we can deduce that a shift s+1 is invalid and that a shift s+2 is potentially valid.

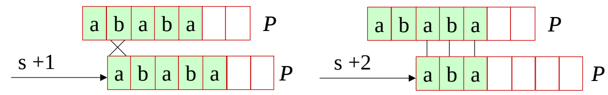


Thus, we can safely transition to state q=3 with a shift $s' = s+2$

- $\pi[5] = 3$ **and** $s' = s + (q - \pi[q])$

The Prefix Function for a Pattern

- Function π can be computed by comparing the pattern against itself

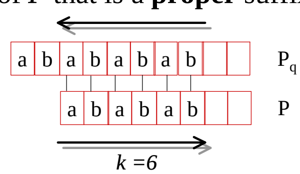


$\Rightarrow$ $\pi[5] = 3$ and $s' = s + (q - \pi[q])$

The Prefix Function for a Pattern

- Formally, π is a function $\{1, 2, \ldots, m\} \rightarrow \{0, 1, \ldots, m-1\}$ such that

  $\pi[q] = \max\{k : k < q \text{ and } P_k \supset P_q\}$
- That is, $\pi[q]$ is the length k of the longest prefix of P that is a **proper** suffix of P_q

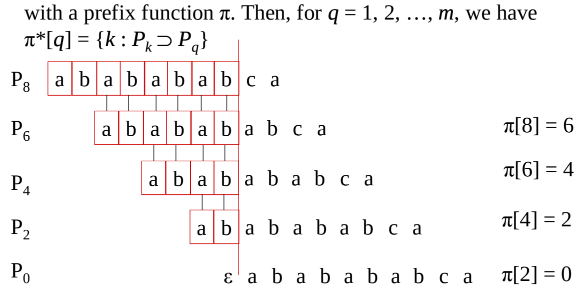


Prefix Function Computation

- We will show that by iterating the prefix function π, we can enumerate all the prefixes P_k that are suffixes of a given prefix P_q.
- Let $\pi^*[q] = \{q, \pi[q], \pi^2[q], \ldots, \pi^t[q]\}$
- where $\pi^i[q]$ is defined in terms of functional composition, so that
  - $\pi^0[q] = q$
  - $\pi^{i+1}[q] = \pi[\pi^i[q]]$ for $i > 0$
- and the sequence in $\pi^*[q]$ stops when $\pi^t[q] = 0$.

Enumerating all the prefixes P_k that are suffixes of a prefix P_q via π*

- *Prefix-function iteration lemma*: Let P be a pattern of length m with a prefix function π. Then, for $q = 1, 2, \ldots, m$, we have $\pi^*[q] = \{k : P_k \supset P_q\}$



$\pi[q] = \max\{k : k < q \text{ and } P_k \supset P_q\}$

Use of π* to Compute the Prefix Function π

Compute-Prefix-Function(*P*)

```
m ← length[P]
π[1] ← 0 ▶ true for any pattern
k ← 0
for q ← 2 to m
    do while k > 0 and P[k + 1] ≠ P[q])
            do k ← π[k]
    if P[k + 1] = P[q]
        then k = k + 1
    π[q] ← k
return π
```


Compute-Prefix-Function(*P*)

```
m ← length[P]
π[1] ← 0 ▶ true for any pattern
k ← 0
for q ← 2 to m
    do while k > 0 and P[k + 1] ≠ P[q])
            do k ← π[k]
    if P[k + 1] = P[q]
        then k = k + 1
    π[q] ← k
return π
```

π[1] ← 0

q = 2 (k = 0)

a	b	a	b	a	b	a	b	c	a

$P[k+1] \neq P[q]$

$k = 0 \Rightarrow$ no smaller prefixes.

$\pi[q] \leftarrow k \Rightarrow \pi[2] \leftarrow 1$

Compute-Prefix-Function(*P*)

```
m ← length[P]
π[1] ← 0 ▶ true for any pattern
k ← 0
for q ← 2 to m
    do while k > 0 and P[k + 1] ≠ P[q])
            do k ← π[k]
    if P[k + 1] = P[q]
        then k = k + 1
    π[q] ← k
return π
```

π[1] ← 0
π[2] ← 0

q = 3 (k = 0)

a	b	a	b	a	b	a	b	c	a

$P[k+1] = P[q]$

$k = k + 1 = 1$ ($P_k \Rightarrow P_1$) .

$\pi[q] \leftarrow k \Rightarrow \pi[3] \leftarrow 1$

P_1 is the longest prefix that is proper suffix of P_3

a	b	a							

a

Compute-Prefix-Function(*P*)

```
m ← length[P]
π[1] ← 0 ▶ true for any pattern
k ← 0
for q ← 2 to m
    do while k > 0 and P[k + 1] ≠ P[q])
            do k ← π[k]
    if P[k + 1] = P[q]
        then k = k + 1
    π[q] ← k
return π
```

π[1] ← 0
π[2] ← 0
π[3] ← 1

q = 4 (k = 1)

a	b	a	b	a	b	a	b	c	a

$P[k+1] = P[q]$

$k = k + 1 = 2$ ($P_k \Rightarrow P_2$) .

$\pi[q] \leftarrow k \Rightarrow \pi[4] \leftarrow 2$

P_2 is the longest prefix that is proper suffix of P_4

a	b	a	b						

a b

Compute-Prefix-Function(*P*)

```
m ← length[P]
π[1] ← 0 ▶ true for any pattern
k ← 0
for q ← 2 to m
    do while k > 0 and P[k + 1] ≠ P[q])
            do k ← π[k]
    if P[k + 1] = P[q]
        then k = k + 1
    π[q] ← k
return π
```

π[1] ← 0
π[2] ← 0
π[3] ← 1
π[4] ← 2

q = 5 (k = 2)

a	b	a	b	a	b	a	b	c	a

$P[k+1] = P[q]$

$k = k + 1 = 3$ ($P_k \Rightarrow P_3$) .

$\pi[q] \leftarrow k \Rightarrow \pi[5] \leftarrow 3$

P_3 is the longest prefix that is proper suffix of P_5

a	b	a	b	a					

a b a

Compute-Prefix-Function(*P*)

```
m ← length[P]
π[1] ← 0 ▶ true for any pattern
k ← 0
for q ← 2 to m
    do while k > 0 and P[k + 1] ≠ P[q])
            do k ← π[k]
    if P[k + 1] = P[q]
        then k = k + 1
    π[q] ← k
return π
```

π[1] ← 0
π[2] ← 0
π[3] ← 1
π[4] ← 2
π[5] ← 3

q = 6 (k = 3)

a	b	a	b	a	b	a	b	c	a

$P[k+1] = P[q]$

$k = k + 1 = 4$ ($P_k \Rightarrow P_4$) .

$\pi[q] \leftarrow k \Rightarrow \pi[6] \leftarrow 4$

P_4 is the longest prefix that is proper suffix of P_6

a	b	a	b	a	b				

a b a b

Compute-Prefix-Function(*P*)

```
m ← length[P]
π[1] ← 0 ▶ true for any pattern
k ← 0
for q ← 2 to m
    do while k > 0 and P[k + 1] ≠ P[q])
            do k ← π[k]
    if P[k + 1] = P[q]
        then k = k + 1
    π[q] ← k
return π
```

π[1] ← 0
π[2] ← 0
π[3] ← 1
π[4] ← 2
π[5] ← 3
π[6] ← 4

q = 7 (k = 4)

a	b	a	b	a	b	a	b	c	a

$P[k+1] = P[q]$

$k = k + 1 = 5$ ($P_k \Rightarrow P_5$) .

$\pi[q] \leftarrow k \Rightarrow \pi[7] \leftarrow 5$

P_5 is the longest prefix that is proper suffix of P_7

a	b	a	b	a	b	a			

a b a b a

Compute-Prefix-Function(*P*)

```
m ← length[P]
π[1] ← 0 ▶ true for any pattern
k ← 0
for q ← 2 to m
    do while k > 0 and P[k + 1] ≠ P[q])
            do k ← π[k]
    if P[k + 1] = P[q]
        then k = k + 1
    π[q] ← k
return π
```

π[1] ← 0
π[2] ← 0
π[3] ← 1
π[4] ← 2
π[5] ← 3
π[6] ← 4
π[7] ← 5

q = 8 (k = 5)

a	b	a	b	a	b	a	b	c	a

$P[k+1] = P[q]$

$k = k + 1 = 6$ ($P_k \Rightarrow P_6$)

$\pi[q] \leftarrow k \Rightarrow \pi[8] \leftarrow 6$

P_6 is the longest prefix that is proper suffix of P_8

a	b	a	b	a	b	a	b		

a b a b a b

```
Compute-Prefix-Function(P)
    m ← length[P]
    π[1] ← 0 ▶ true for any pattern
    k ← 0
    for q ← 2 to m
        do while k > 0 and P[k + 1] ≠ P[q])
                do k ← π[k]
        if P[k + 1] = P[q]
          then k = k + 1
        π[q] ← k
    return π
```

$\pi[1] \leftarrow 0$ $\pi[6] \leftarrow 4$
$\pi[2] \leftarrow 0$ $\pi[7] \leftarrow 5$
$\pi[3] \leftarrow 1$ $\pi[8] \leftarrow 6$
$\pi[4] \leftarrow 2$
$\pi[5] \leftarrow 3$

$\boxed{q = 9}$ (k = 6)

a	b	a	b	a	b	a	b	c	a

$P[k + 1] \neq P[q]$

$k = \pi[k] \Rightarrow k \leftarrow \pi[6] = 4$

a	b	a	b	a	b	a	b	c	
		a	b	a	b	a	b	a	


```
Compute-Prefix-Function(P)
    m ← length[P]
    π[1] ← 0 ▶ true for any pattern
    k ← 0
    for q ← 2 to m
        do while k > 0 and P[k + 1] ≠ P[q])
                do k ← π[k]
        if P[k + 1] = P[q]
          then k = k + 1
        π[q] ← k
    return π
```

$\pi[1] \leftarrow 0$ $\pi[6] \leftarrow 4$
$\pi[2] \leftarrow 0$ $\pi[7] \leftarrow 5$
$\pi[3] \leftarrow 1$ $\pi[8] \leftarrow 6$
$\pi[4] \leftarrow 2$
$\pi[5] \leftarrow 3$

$\boxed{q = 9}$ (k = 4)

a	b	a	b	a	b	a	b	c	a

$P[k + 1] \neq P[q]$

$k = \pi[k] \Rightarrow k \leftarrow \pi[4] = 2$

a	b	a	b	a	b	a	b	c	
				a	b	a	b	a	


```
Compute-Prefix-Function(P)
    m ← length[P]
    π[1] ← 0 ▶ true for any pattern
    k ← 0
    for q ← 2 to m
        do while k > 0 and P[k + 1] ≠ P[q])
                do k ← π[k]
        if P[k + 1] = P[q]
          then k = k + 1
        π[q] ← k
    return π
```

$\pi[1] \leftarrow 0$ $\pi[6] \leftarrow 4$
$\pi[2] \leftarrow 0$ $\pi[7] \leftarrow 5$
$\pi[3] \leftarrow 1$ $\pi[8] \leftarrow 6$
$\pi[4] \leftarrow 2$
$\pi[5] \leftarrow 3$

$\boxed{q = 9}$ (k = 2)

a	b	a	b	a	b	a	b	c	a

$P[k + 1] \neq P[q]$

$k = \pi[k] \Rightarrow k \leftarrow \pi[2] = 0$

a	b	a	b	a	b	a	b	c	
						a	b	a	


```
Compute-Prefix-Function(P)
    m ← length[P]
    π[1] ← 0 ▶ true for any pattern
    k ← 0
    for q ← 2 to m
        do while k > 0 and P[k + 1] ≠ P[q])
                do k ← π[k]
        if P[k + 1] = P[q]
          then k = k + 1
        π[q] ← k
    return π
```

$\pi[1] \leftarrow 0$ $\pi[6] \leftarrow 4$
$\pi[2] \leftarrow 0$ $\pi[7] \leftarrow 5$
$\pi[3] \leftarrow 1$ $\pi[8] \leftarrow 6$
$\pi[4] \leftarrow 2$
$\pi[5] \leftarrow 3$

$\boxed{q = 9}$ (k = 0)

a	b	a	b	a	b	a	b	c	a

$P[k + 1] \neq P[q]$

$\pi[q] \leftarrow k \Rightarrow \pi[9] \leftarrow 0$

a	b	a	b	a	b	a	b	c	
								a	


```
Compute-Prefix-Function(P)
    m ← length[P]
    π[1] ← 0 ▶ true for any pattern
    k ← 0
    for q ← 2 to m
        do while k > 0 and P[k + 1] ≠ P[q])
                do k ← π[k]
        if P[k + 1] = P[q]
          then k = k + 1
        π[q] ← k
    return π
```

$\pi[1] \leftarrow 0$ $\pi[6] \leftarrow 4$
$\pi[2] \leftarrow 0$ $\pi[7] \leftarrow 5$
$\pi[3] \leftarrow 1$ $\pi[8] \leftarrow 6$
$\pi[4] \leftarrow 2$ $\pi[9] \leftarrow 0$
$\pi[5] \leftarrow 3$

$\boxed{q = 10}$ (k = 0)

a	b	a	b	a	b	a	b	c	a

$P[k + 1] = P[q]$

$k = k + 1 = 1\ (P_k \Longrightarrow P_1)$

$\boxed{\pi[q] \leftarrow k \Rightarrow \pi[10] \leftarrow 1}$

a	b	a	b	a	b	a	b	c	a
									a

The KMP Algorithm

```
KMP-Matcher(T, P)
    n ← length[T]
    m ← length[P]
    π ← Compute-Prefix-Function(P)
    q ← 0 ▶ current position in P
    for i ← 1 to n ▶ current position in T
        do while q > 0 and P[q + 1] ≠ T[i])
                do q ← π[q]
            if P[k + 1] = T[i]
                then q = q + 1
            if q = m
                then print "Pattern occurs with shift" i – m
                     q ← π[q]
```