

**Курс «Алгоритмы и алгоритмические языки»  
1 семестр 2013/2014**

**Лекция 17**

## **Формальная постановка задачи поиска по образцу**

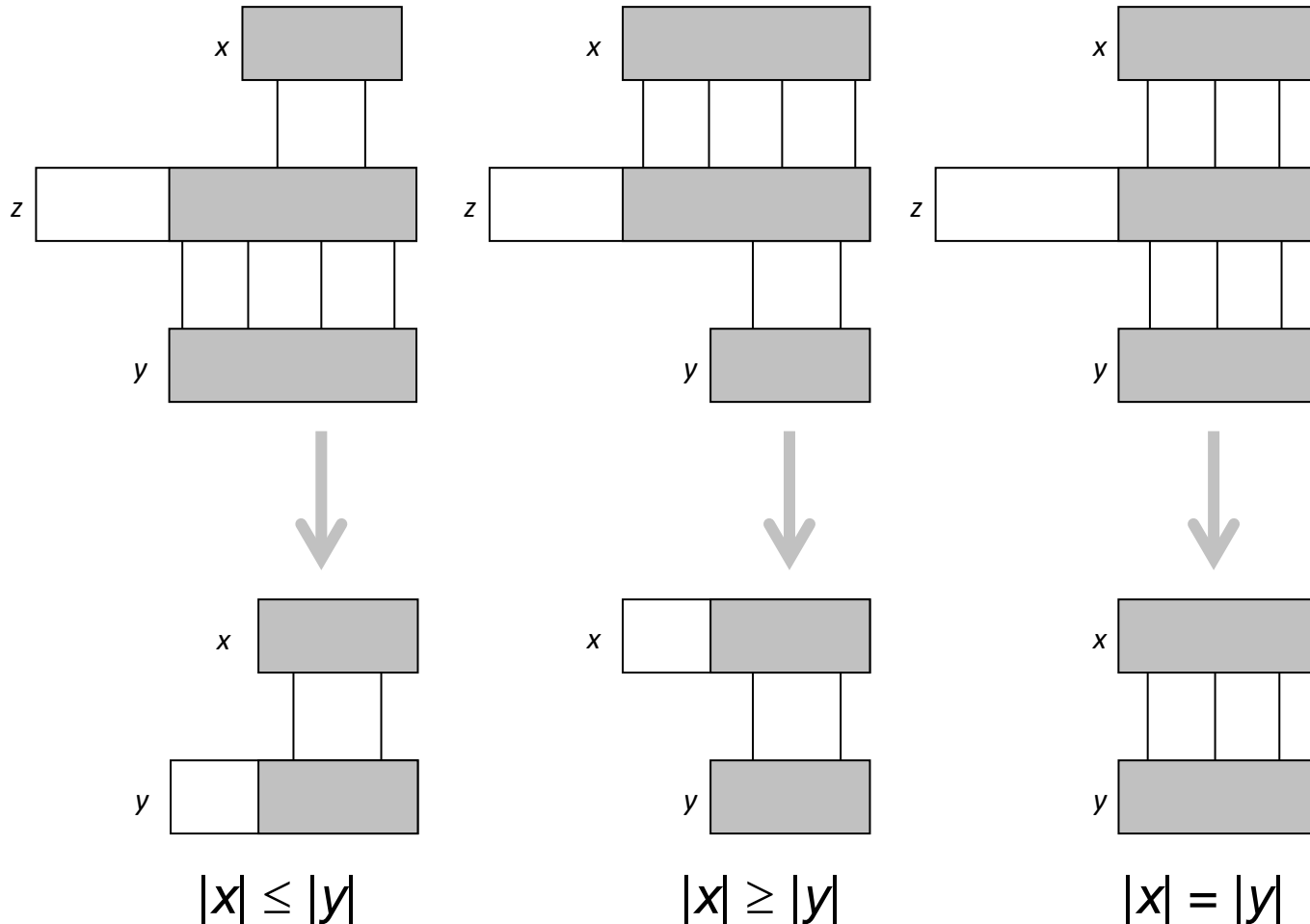
- ◇ Даны *текст* – массив  $T[N]$  длины  $N$  и *образец* – массив  $P[m]$  длины  $m \leq N$ , где значениями элементов массивов  $T$  и  $P$  являются символы некоторого *алфавита*  $A$ .
- ◇ Говорят, что образец  $P$  входит в текст  $T$  со сдвигом  $s$ , если  $0 \leq s \leq N - m$  и для всех  $i = 1, 2, \dots, m$   $T[s + i] = P[i]$ .
- ◇ Сдвиг  $s(T, P)$  называется *допустимым*, если  $P$  входит в  $T$  со сдвигом  $s = s(T, P)$  и *недопустимым* в противном случае.
- ◇ **Задача поиска подстрок** состоит в нахождении множества допустимых сдвигов  $s(T, P)$  для заданного текста  $T$  и образца  $P$ .

## Формальная постановка задачи поиска по образцу

- ◇ **Терминология.** Пусть строки  $x, y, w \in A^*$ ,  $\varepsilon \in A^*$  - пустая строка;  
 $|x|$  - длина строки  $x$ ;  
 $xy$  – *конкатенация* строк  $x$  и  $y$ ;  $|xy| = |x| + |y|$ ;  
 $x = wy \rightarrow w$  – *префикс* (начало)  $x$  (обозначение  $w \prec x$  );  
 $x = yw \rightarrow w$  – *суффикс* (конец)  $x$  (обозначение  $w \succ x$  );  
если  $w$  – префикс или суффикс  $x$ , то  $|w| \leq |x|$ ;  
отношения префикса и суффикса *транзитивны*.  
Для любых  $x, y \in A^*$  и любого  $a \in A$  соотношения  $x \succ y$   
и  $xa \succ ya$  *равносильны*.
- ◇ Если  $S = S[r]$  – строка длины  $r$ , то ее префикс длины  $k$ ,  $k \leq r$  будет обозначаться  $S_k = S[k]$ ; ясно, что  $S_0 = \varepsilon$ ,  $S_r = S$ .

# Лемма (о двух суффиксах)

- ◇ Пусть  $x$ ,  $y$  и  $z$  – строки, для которых  $x \succ z$  и  $y \succ z$ .  
Тогда если  $|x| \leq |y|$ , то  $x \succ y$ , если  $|x| \geq |y|$ , то  $y \succ x$ ,  
если  $|x| = |y|$ , то  $x = y$ .



## Простой алгоритм

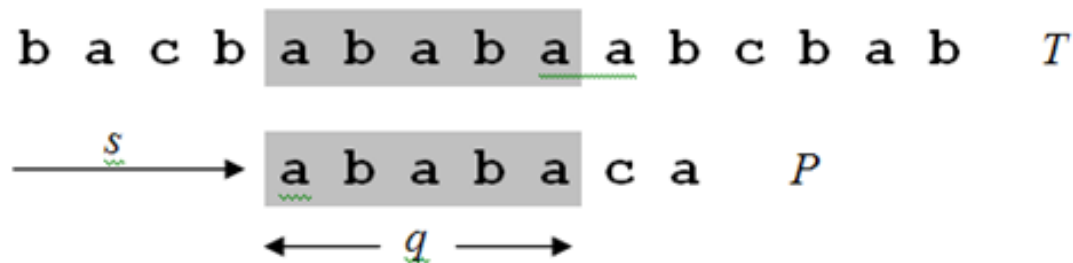
- ◇ Проверка совмещения двух строк: посимвольное сравнение слева направо, которое прекращается (с отрицательным результатом) при первом же расхождении.
- ◇ Оценка скорости сравнения строк  $x$  и  $y - \Theta(t + 1)$ , где  $t$  – длина наибольшего общего префикса строк  $x$  и  $y$ .

```
for (s = 0; s <= n - m; s++) {  
    for (i = 0; i < m && P[i] == T[s + i]; i++)  
        ;  
    if (i == m)  
        printf ("%d\n", s);  
}
```

- ◇ Время работы в худшем случае  $\Theta((n - m + 1) \cdot m) \sim \Theta(n^2)$ .  
*Причина:* информация о тексте  $T$ , полученная при проверке данного сдвига  $s$ , никак не используется при проверке следующих сдвигов. Например, если для образца **dddc** сдвиг  $s = 0$  допустим, то сдвиги  $s = 1, 2, 3$ , недопустимы, так как **T[3] == c**.

# Алгоритм Кнута – Морриса – Пратта. Идея

- ◇ Префикс-функция, ассоциированная с образцом  $P$ , показывает, где в строке  $P$  повторно встречаются различные префиксы этой строки. Если это известно, можно не проверять заведомо недопустимые сдвиги.
- ◇ **Пример.** Пусть ищутся вхождения образца  $P = \mathbf{a b a b a c a}$  в текст  $T$ . Пусть для некоторого сдвига  $s$  оказалось, что первые  $q$  символов образца совпадают с символами текста. Значит, символы текста от  $T[s+1]$  до  $T[s+q]$  известны, что позволяет заключить, что некоторые сдвиги заведомо недопустимы.



## Алгоритм Кнута – Морриса – Пратта. Идея

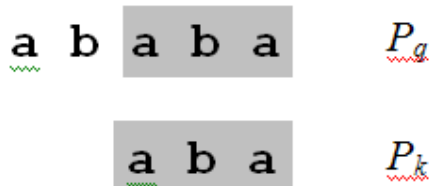
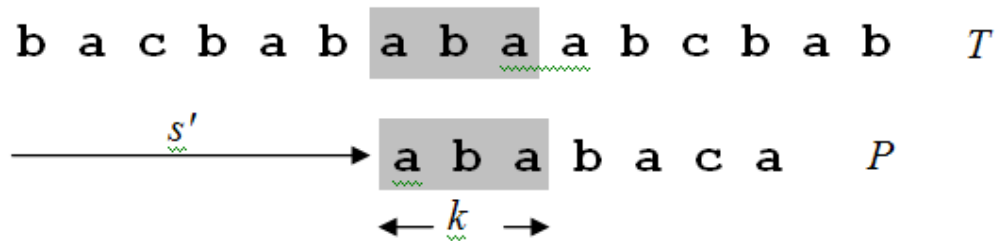
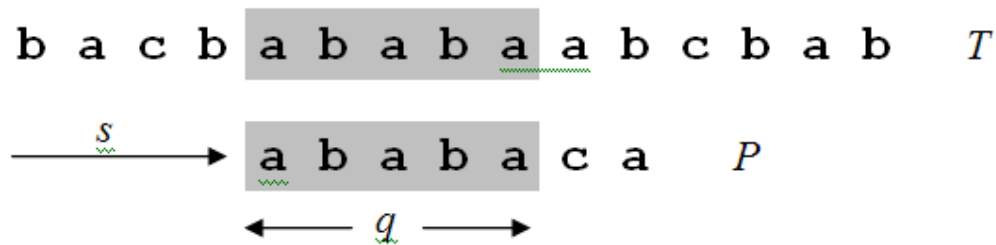
- ◇ Пусть  $P[1..q] = T[s+1..s+q]$ ; каково минимальное значение сдвига  $s' > s$ , для которого  $P[1..k] = T[s'+1..s'+k]$ , где  $s'+k = s+q$ ?
  - ◆ Число  $s'$  - минимальное значение сдвига, большего  $s$ , которое совместимо с тем, что  $T[s+1..s+q] = P[1..q]$ . Следовательно, значения сдвигов, меньшие  $s'$ , проверять не нужно. Лучше всего, когда  $s' = s+q$ , так как в этом случае не нужно рассматривать сдвиги  $s+q-1, s+q-2, \dots, s+1$ . Кроме того, при проверке нового сдвига  $s'$  можно не рассматривать первые его  $k$  символов образца: они заведомо совпадут.
- ◇ Чтобы найти  $s'$ , достаточно знать образец  $P$  и число  $q$ :  $T[s'+1..s'+k]$  – суффикс  $P_q$ , поэтому  $k$  – это наибольшее число, для которого  $P_k$  является суффиксом  $P_q$ . Зная  $k$  (число символов, заведомо совпадающих при проверке нового сдвига  $s'$ ), можно вычислить  $s'$  по формуле  $s' = s + (q - k)$ .

# Алгоритм Кнута – Морриса – Пратта. Префикс-функция

♦ **Определение.** Префикс-функцией, ассоциированной со строкой  $P[1..m]$ , называется функция  $\pi: \{1, 2, \dots, m\} \rightarrow \{0, 1, \dots, m-1\}$ , определенная следующим образом:

$$\pi[q] = \max\{k: k < q \ \& \ P_k \succcurlyeq P_q\}$$

Иными словами,  $\pi[q]$  – длина наибольшего префикса  $P$ , являющегося суффиксом  $P_q$ .





# Алгоритм Кнута – Морриса – Пратта. Префикс-функция

```
void prefix_func (char *pat, int *pi, int m) {
    int k, pi;

    /* Считаем, что pat и pi нумеруются от 1 */
    pi[1] = 0; k = 0;
    for (q = 2; q <= m; q++) {
        while (k > 0 && pat[k + 1] != pat[q])
            k = pi[k];
        if (pat[k + 1] == pat[q])
            k++;
        pi[q] = k;
    }
}
```

# Алгоритм Кнута – Морриса – Пратта. Префикс-функция

◇ **Лемма 1.** Обозначим  $\pi^*[q] = \{q, \pi[q], \pi^2[q], \dots, \pi^t[q]\}$ , где  $\pi^i[q]$  есть  $i$ -я итерация префикс-функции,  $\pi^t[q] = 0$ . Пусть  $P$  – строка длины  $m$  с префикс-функцией  $\pi$ . Тогда для всех  $q = 1, 2, \dots, m$  имеем  $\pi^*[q] = \{k : P_k \succ P_q\}$ .

◇ Лемма показывает, что при помощи итерирования префикс-функции можно для данного  $q$  найти все такие  $k$ , что  $P_k$  является суффиксом  $P_q$ .

◇ Доказательство.

(1) Покажем, что если  $i$  принадлежит  $\pi^*[q]$ , то  $P_i$  является суффиксом  $P_q$ .

Действительно,  $P_{\pi[i]} \succ P_i$  по определению префикс-функции, так что каждый следующий член последовательности  $P_i, P_{\pi[i]}, P_{\pi[\pi[i]]}, \dots$  является суффиксом всех предыдущих.

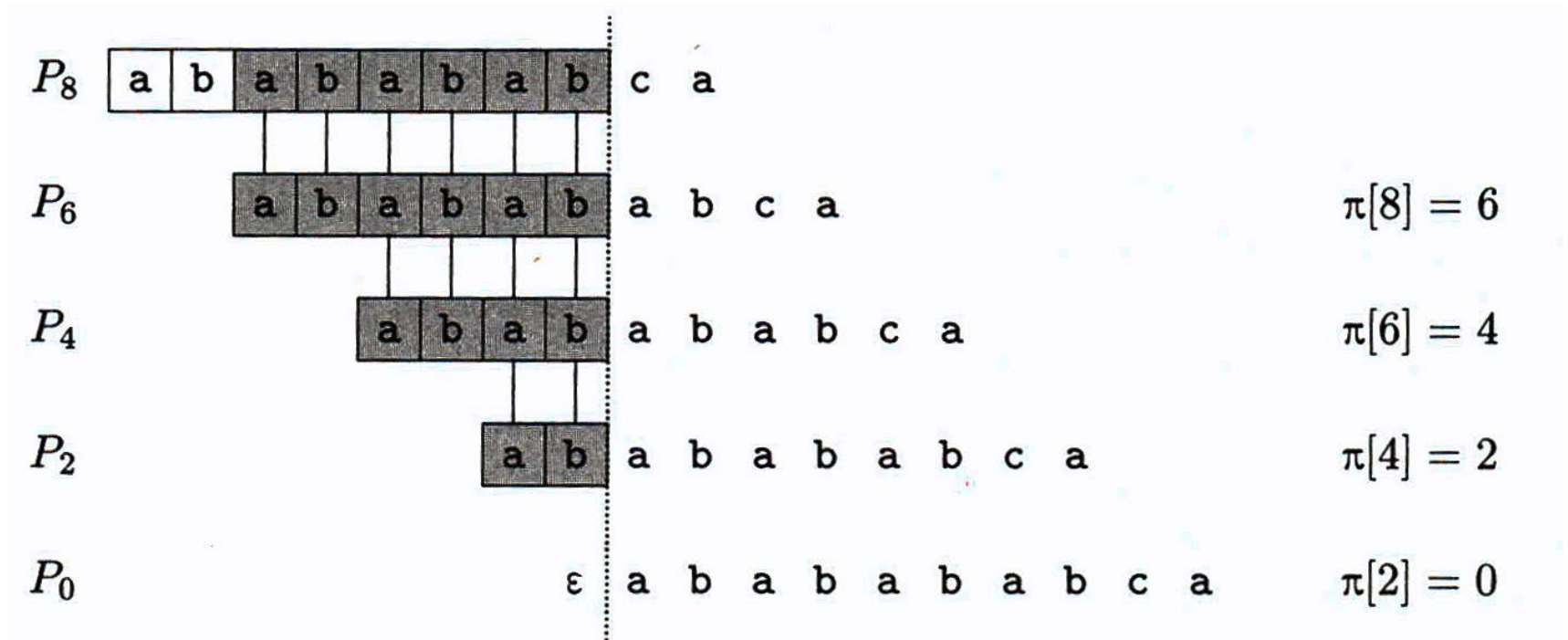
# Алгоритм Кнута – Морриса – Пратта. Префикс-функция

◇ Доказательство.

- (2) Покажем, что наоборот, если  $P_i$  является суффиксом  $P_q$ , то  $i$  принадлежит  $\pi^*[q]$ .
- ◇ Расположим все  $P_i$ , являющиеся суффиксами  $P_q$ , в порядке уменьшения  $i$  (длины):  $P_{i_1}, P_{i_2}, \dots$
  - ◇ Покажем по индукции, что  $P_{i_k} = \pi^k[q]$ .
  - ◇ База индукции ( $k=1$ ): для максимального префикса  $P_i$ , являющегося суффиксом  $P_q$ , по определению  $i = \pi[q]$ .
  - ◇ Шаг индукции: если  $P_{i_k} = \pi^k[q]$ , то по определению  $j = \pi[\pi^k[q]]$  соответствует максимальный префикс  $P_j$ , который является суффиксом  $P_{i_k}$ . Обе строки  $P_j$  и  $P_{i_k}$  есть суффиксы  $P_q$  по построению. Таким максимальным префиксом из оставшихся  $P_{i_{k+1}}, P_{i_{k+2}}, \dots$  по построению является префикс  $P_{i_{k+1}}$ , то есть  $j = i_{k+1}$ .
  - ◇ (2) можно доказать и от противного: для наибольшего числа  $j$  такого, что  $P_j \succ P_q$ , но  $j$  не входит в  $\pi^*[q]$ , определение префикс-функции нарушается

# Алгоритм Кнута – Морриса – Пратта. Префикс-функция

$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1



$$\pi^*[8] = \{8, 6, 4, 2, 0\}$$

## Алгоритм Кнута – Морриса – Пратта. Префикс-функция

- ◇ **Лемма 2.** Пусть  $P$  – строка длины  $m$  с префикс-функцией  $\pi$ . Тогда для всех  $q = 1, 2, \dots, m$ , для которых  $\pi[q] > 0$ , имеем  $\pi[q] - 1 \in \pi^*[q - 1]$ .
- ◇ Доказательство.
  - ◆ Если  $k = \pi[q] > 0$ , то  $P_k$  является суффиксом  $P_q$  по определению префикс-функции.
  - ◆ Следовательно,  $P_{k-1}$  является суффиксом  $P_{q-1}$ .
  - ◆ Тогда по Лемме 1  $k - 1 \in \pi^*[q - 1]$ , т.е.  
 $\pi[q] - 1 \in \pi^*[q - 1]$ .
- ◇ Определим множества  $E_{q-1}$  как  
 $E_{q-1} = \{k : k \in \pi^*[q - 1] \text{ и } P[k + 1] = P[q]\}$ .  
Множество  $E_{q-1}$  состоит из таких  $k$ , что  $P_k$  является суффиксом  $P_{q-1}$ , и за ними идут одинаковые буквы  $P[k+1]$  и  $P[q]$ .  
Из определения вытекает, что  $P_{k+1}$  есть суффикс  $P_q$ .

# Алгоритм Кнута – Морриса – Пратта. Префикс-функция

- ◇ **Следствие 1.** Пусть  $P$  – строка длины  $m$  с префикс-функцией  $\pi$ . Тогда для всех  $q = 2, 3, \dots, m$

$$\pi[q] = \begin{cases} 0, & \text{если } E_{q-1} \text{ пусто;} \\ 1 + \max\{k \in E_{q-1}\}, & \text{если } E_{q-1} \text{ не пусто.} \end{cases}$$

- ◇ Доказательство.

- ◆ Если  $r = \pi[q] \geq 1$ , то  $P[r] = P[q]$  и по Лемме 2  $r - 1 = \pi[q] - 1 \in \pi^*[q - 1]$ .
- ◆ Т.к.  $P[r] = P[q]$ , то  $P[(r-1)+1] = P[q]$ .
- ◆ Поэтому  $r - 1 \in E_{q-1}$  по определению  $E_{q-1}$  и из  $\pi[q] \geq 1$  следует непустота  $E_{q-1}$ .
- ◆ Следовательно, если  $E_{q-1}$  пусто, то  $\pi[q] = 0$ .
- ◆ Если  $k \in E_{q-1}$ , то  $P_{k+1}$  есть суффикс  $P_q$  (из определения), следовательно,  $\pi[q] \geq k + 1$  и  $\pi[q] \geq 1 + \max\{k \in E_{q-1}\}$ .
- ◆ То есть, если  $E_{q-1}$  не пусто, то префикс-функция положительна. Но тогда  $\pi[q] - 1 \in E_{q-1}$ ,  $\pi[q] - 1$  не больше максимума из  $E_{q-1}$ , т.е.  $\pi[q] \leq 1 + \max\{k \in E_{q-1}\}$ .

## Алгоритм Кнута – Морриса – Пратта. Префикс-функция

```
1 void prefix_func (char *pat, int *pi, int m) {
2     int k, pi;
3
4     /* Считаем, что pat и pi нумеруются от 1 */
5     pi[1] = 0; k = 0;
6     for (q = 2; q <= m; q++) {
7         while (k > 0 && pat[k + 1] != pat[q])
8             k = pi[k];
9         if (pat[k + 1] == pat[q])
10            k++;
11        pi[q] = k;
12    }
13 }
```

# Алгоритм Кнута – Морриса – Пратта. Префикс-функция

◆ **Теорема 1.** Функция `prefix_func` правильно вычисляет префикс-функцию  $\pi$ .

◆ Доказательство.

◆ Покажем, что при входе в цикл функции  $k = \pi[q-1]$ .

◆ База индукции.

При  $q = 2$   $k = 0$ ,  $pi[q-1] = pi[1] = 0$ .

◆ Шаг индукции.

Пусть при входе в цикл функции  $k = \pi[q-1]$ .

Код на строках 7-8

```
while (k > 0 && pat[k + 1] != pat[q])  
    k = pi[k];
```

находит наибольший элемент  $E_{q-1}$  (т.к. цикл перебирает в порядке убывания элементы из  $\pi^*[q-1]$  и для каждого проверяет равенство `pat[k + 1] != pat[q]`)).



# Алгоритм Кнута – Морриса – Пратта. Префикс-функция

◆ **Теорема 1.** Функция `prefix_func` правильно вычисляет префикс-функцию  $\pi$ .

◆ Доказательство.

◆ После выхода из цикла на строках 7-8

```
while (k > 0 && pat[k + 1] != pat[q])  
    k = pi[k];
```

1) если `pat[k + 1] == pat[q]`, то выполняется код на строке 10:

```
k++;
```

что из Следствия 1 дает нам  $\pi[q]$ .

2) если `pat[k + 1] != pat[q]`, то `k == 0`, множество  $E_{q-1}$  пусто и  $\pi[q] = 0$ .

## **Алгоритм Кнута – Морриса – Пратта. Функция kmp**

```
void kmp (char *text, char *pat, int m, int n) {
    int q;
    int pi[m + 1]; /* VLA-массив */
    /* Через alloca: int *pi = alloca ((m + 1) * sizeof (int)); */
    /* Считаем, что pat и text нумеруются от 1 */
    prefix_func (pat, pi, m);
    q = 0;
    for (i = 1; i <= n; i++) {
        while (q > 0 && pat[q + 1] != text[i])
            q = pi[q];
        if (pat[q + 1] == text[i])
            q++;
        if (q == m) {
            printf ("образец входит со сдвигом %d\n", i - m);
            q = pi[q];
        }
    }
}
```

# Алгоритм Кнута – Морриса – Пратта. Функция $kmp$

- ◇ Алгоритм КМП для подстроки  $P$  и текста  $T$  эквивалентен вычислению префикс-функции для строки  $Q = P\#T$ , где  $\#$  – символ, заведомо не встречающийся в обеих строках
  - ◆ Длина максимального префикса  $Q$ , являющегося её суффиксом (т.е. значение префикс-функции), не превосходит длины  $P$
  - ◆ Допустимый сдвиг обнаруживается в тот момент, когда очередное вычисленное значение префикс-функции совпадает с длиной подстроки  $P$  (условие `if (q == m)`)
  - ◆ В явном виде объединенная строка не строится!
- ◇ **Теорема 2.** Функция  $kmp$  работает правильно.
  - ◆ Формальное доказательство осуществляется по аналогии с доказательством Теоремы 1, где множества, подобные  $E_{q-1}$ , строятся для строки-текста, а не строки-образца.
- ◇ Свойства префикс-функции часто используются и в других задачах (кроме поиска подстроки в строке)
  - ◆ Полезной оказывается Лемма 1: итерированием префикс-функции можно найти все префиксы строки, являющиеся ее суффиксами

# Алгоритм Кнута – Морриса – Пратта. Время работы

- ◆ Функция **prefix\_func** выполняет  $\leq (m - 1)$  итераций цикла **for**. Стоимость каждой итерации можно считать равной  $O(1)$ , а стоимость всей процедуры  $O(m)$ .
  - ◆ Каждая итерация цикла **while** (строки 7-8) уменьшает **k**
  - ◆ Увеличивается **k** только в строке 10 не более одного раза на итерацию цикла **for** (строки 6-11)
  - ◆ Следовательно, операций уменьшения не больше, чем итераций цикла **for**, то есть  $\leq (m - 1)$  на весь цикл и  $O(1)$  на итерацию в среднем
- ◆ Аналогично, функция **kmp** выполняет  $\leq (n - 1)$  итераций, и ее стоимость (без учета вызова **prefix\_func**) есть  $O(n)$ . Следовательно, время выполнения всей процедуры  $O(m + n)$ .