Лекция 20 Поиск подстрок по образцу

20.1. ***Что такое поиск подстроки по образцу***?

20.1.1. ***Неформальная постановка задачи поиска по образцу***.

20.1.2. ***Формальная постановка задачи поиска по образцу***.

Даны *текст* – массив $T[N]$ длины N и *образец* – массив $P[m]$ длины $m \le N$, где значениями элементов массивов T и P являются символы некоторого *алфавита* A. Говорят, что образец P входит в текст T со сдвигом s (или, что то же самое, с позиции $s + 1$), если $0 \le s \le N - m$ и для всех $i = 0, 1, \ldots, m - 1$ $T[s + i] = P[i]$. 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.

20.1.3. ***Терминология***.

Пусть строки $x, y, w \in A^*$, $\varepsilon \in A^*$ - пустая строка: $|x|$ - длина строки x; xy – *конкатенация* строк x и y; $|xy| = |x| + |y|$; $x = wy$ w – *префикс* (начало) x (обозначение $w \sqsubset x$); $x = yw$ w – *суффикс* (конец) x (обозначение $w \hbar x$); если w – префикс или суффикс x, то $|w| \le |x|$; отношения $\sqsubset$ и $\hbar$ *транзитивны*. Для любых $x, y \in A^*$ и любого $a \in A$ соотношения $x \hbar y$ и $xa \hbar ya$ *равносильны*.

Если $S = S[r]$ – строка длины r, то ее префикс длины k, $k \le r$ будет обозначаться $S_k = S[k]$; ясно, что $S_0 = \varepsilon$, $S_r = S$

20.2. ***Лемма*** (***о двух суффиксах***).

Пусть x, y и z – строки, для которых $x \hbar z$ и $y \hbar z$. Тогда если $|x| \le |y|$, то $x \hbar y$, если $|x| \ge |y|$, то $y \hbar x$, если $|x| = |y|$, то $x = y$.

Доказательство.

См. рисунок.

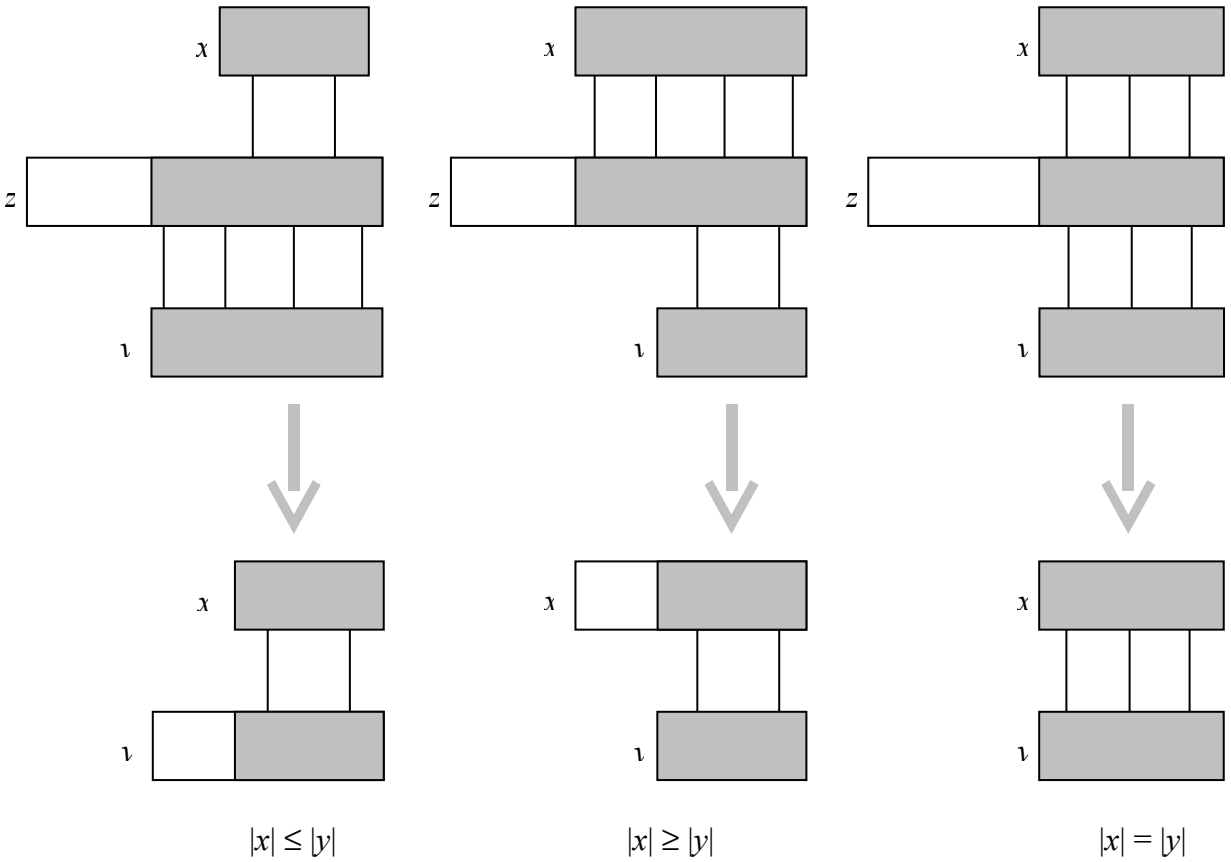

Рис. 1. Лемма о двух суффиксах.

20.3. ***Простейший алгоритм***.

20.3.1. 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.

20.3.2. ***Алгоритм***: передвигаем P вдоль T, каждый раз увеличивая сдвиг на 1 и сравнивая P и соответствующую часть T:

```
i = 0; while(P[i] == T[s + i]) i++; if(i == m) print("...");
```

20.3.3. Основной цикл:

```
for(s = 0; s <= n - m; s++) {
        i = 0;
        while(P[i] == T[s + i]) i++;
        if(i == m) printf(" ");
}
```

20.3.4. 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**.

20.4. ***Алгоритм Кнута – Морриса – Пратта***.

20.4.1. Работает за время Θ(*n* + *m*).

20.4.2. 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.

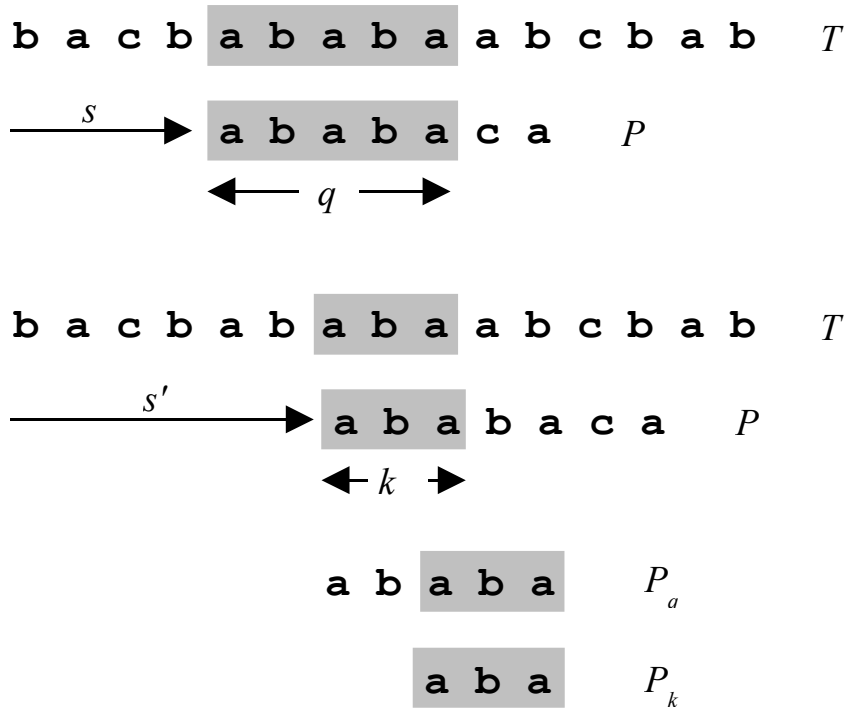

Рис. 2. Префикс-функция:

20.4.3. ***Пример***.

Пусть ищутся вхождения образца *P* = **a b a b a c a** в текст *T* (см. рисунок). Пусть для некоторого сдвига *s* оказалось, что первые *q* символов образца совпадают с символами текста (на рисунке *q* = 5). Значит, символы текста от *T*[*s*+1] до *T*[*s*+*q*] известны, что позволяет заключить, что некоторые сдвиги заведомо недопустимы (например, на рисунке недопустим сдвиг *s* + 1, так как **a** будет напротив **b**, а при сдвиге *s* + 2 – совпадение первых трех символов, и его без проверок отбросить нельзя).

20.4.4. ***Вопрос***.

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 $s+q, s+q, ..., s+q-1$. 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 $s' = s + (q - k)$.

20.4.5. ***Определение***.

Префикс-функцией, ассоциированной со строкой $P[1..m]$, называется функция π: $\{1,2, \ldots, m\} \to \{0,1, \ldots, m-1\}$, определенная следующим образом:

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

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

На рисунке – префикс-функция для строки $P =$ **a b a b a c a**.

20.4.6. ***Алгоритм***.

```
#inciude <stdio.h>
#inciude <string.h>

void KMP-Matcher(char *T, char *P) {
   int m, n, q, pi;
   n = strlen(T);
   m = strlen(P);
   pi = PrefixFunc(P);
   q = 0;
   for(i = 0; i < n; i++) {
     while(q > 0 && P[q + 1] != T[i]) q += pi;
     if(P[q + 1] == T[i]) q += 1;
     if(q == m) {
           printf("образец входит со сдвигом %d" i – m);
           q = pi;
     }


int PrefixFunc(char *P) {
```

```
    int m, k, pi;
    m = strlen(P);
    pi = 0; k = 0;
    for(q = 1; q < m; q++) {
      while(k > 0 && P[k + 1] != P[q]) k++;
      if(P[k + 1] == P[q]) k += 1;
    }
    return k;
}
```

20.4.7. ***Время работы***.

PrefixFunc выполняет ≤ (m – 1) итераций цикла **for** . Стоимость каждой итерации $O(1)$, а стоимость всей процедуры $O(m)$. Аналогично **KMP-Matcher** выполняет ≤ (n – 1) итераций, и ее стоимость $O(n)$. Следовательно, время выполнения всей процедуры $O(m + n)$

20.4.8. ***Обоснование алгоритма***.