СУФФИКСНЫЕ СТРУКТУРЫ ДАННЫХ

Антон Ахи

Сергей Поромов

2009г.

Введение

- Существует множество задач на строках. Многие из них имеют прикладное значение (например в генетике, обработке текстов). Для эффективного решения большого числа строковых задач используют следующие структуры данных:
  - Суффиксное дерево
  - Суффиксный массив
  - Суффиксный автомат

Задача о подстроке

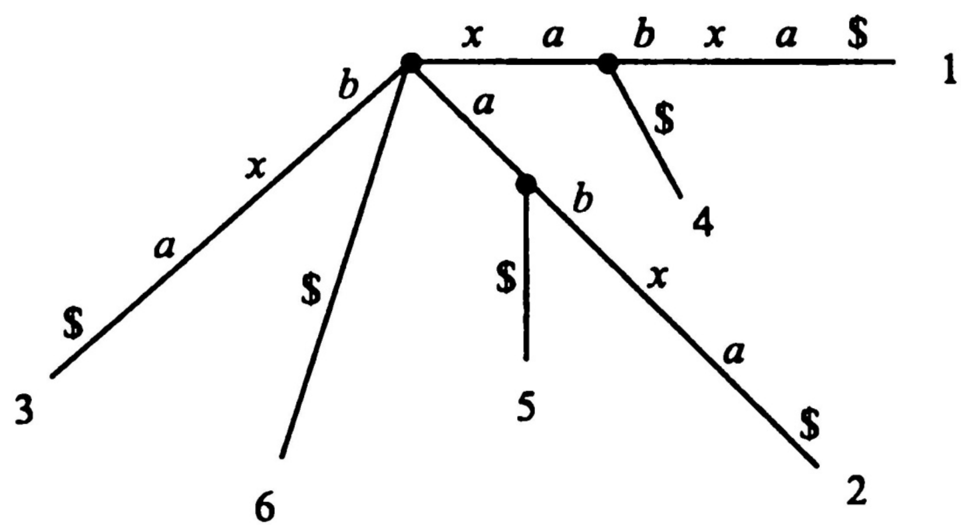
- Наиболее известной является задача о подстроке – нахождение одного или всех вхождений одной строки (длины m, называемой образцом) в тексте (строка длины n). Такой поиск можно осуществлять алгоритмом КМП, однако при множественном поиске такой метод теряет свою эффективность.
- Рассматриваемые структуры данных требуют препроцессинг за O(n). Однако затем поиск каждого образца будет занимать O(m).

Суффиксное дерево

- Суффиксное дерево является, пожалуй, наиболее мощной структурой из представленных. Существует два алгоритма построения суффиксного дерева за линейное время:
    - алгоритм Укконена
    - алгоритм МакКрейта
- Однако оба этих алгоритма достаточно сложны и не будут рассмотрены.

Суффиксное дерево

- Суффиксное дерево – сжатый бор, содержащий все суффиксы данной строки.
- Содержит O(n) вершин (не более 2 n –1).
- Суффиксное дерево для строки «xabxa$»:

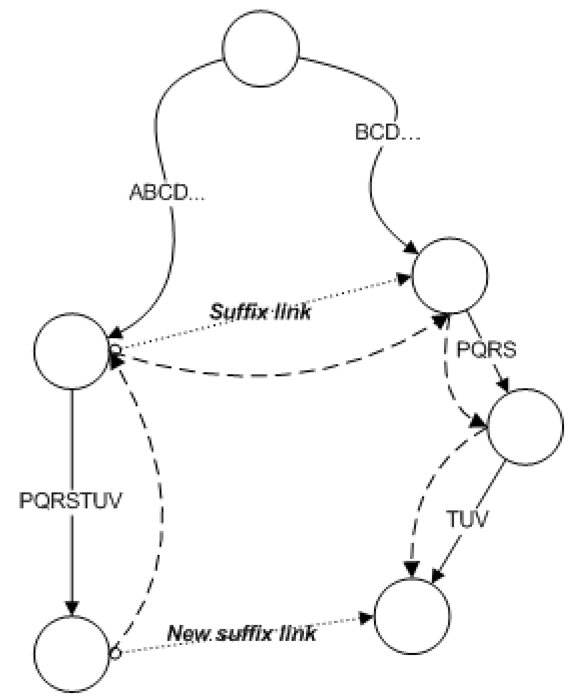


Суффиксные ссылки

- Для многих алгоритмов на суффиксных деревьях и автоматах необходимо знание суффиксных ссылок.
- Суффиксная ссылка – ссылка из вершины, соответствующей слову s, в вершину, соответствующую слову s без первого символа.
- Суффиксные ссылки для суффиксного дерева легко построить за O(n).

Нахождение суффиксных ссылок

- Находятся сверху вниз (обходом в ширину).
- Для вершины (v) – идем в родителя (u) и из него по его суффиксной ссылке в (u’).
- Далее из этой вершины (u’) идем по тем буквам, что написаны на (u-v).



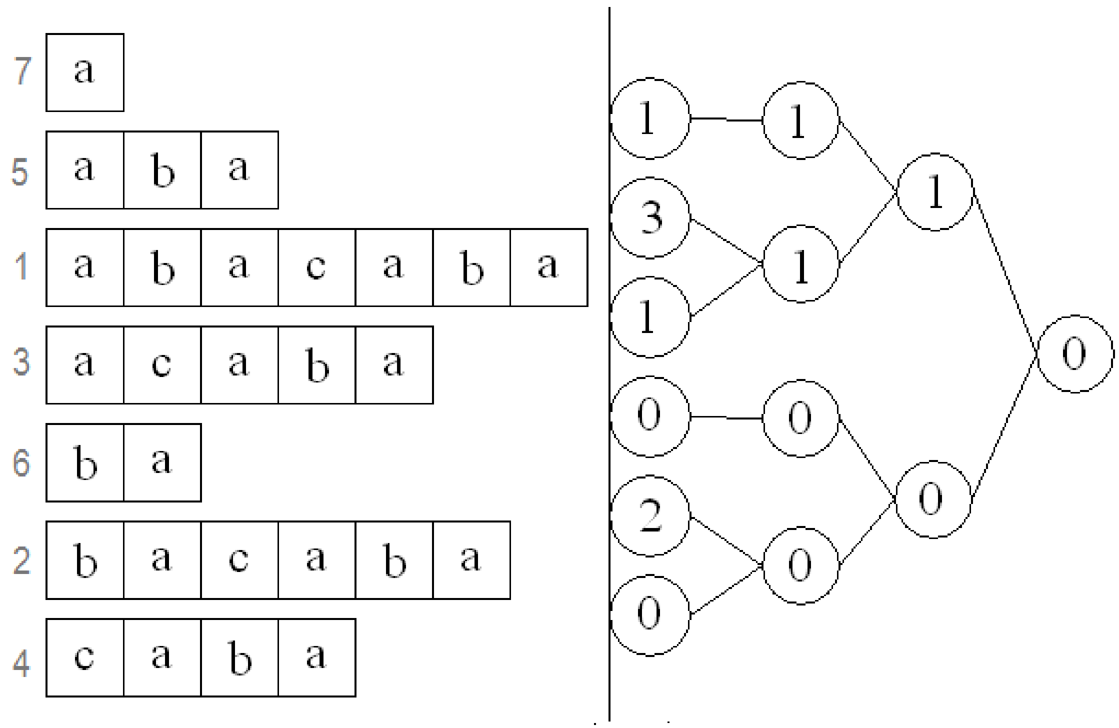
Задачи для суффиксного дерева

- Проверка того, является ли строка подстрокой другой
- Нахождение наибольшей общей подстроки
- Подсчет количества различных подстрок
- Подсчет суммарной длины различных подстрок
- Нахождение наибольшей подстроки-палиндрома
- Нахождение наибольшей повторяющейся подстроки

Суффиксный массив

- Массив номеров суффиксов, упорядоченных в лексикографическом порядке
- Массив для RMQ на LCP соседних суффиксов

Пример суффиксного массива для строки « abacaba»:



Суффиксный массив

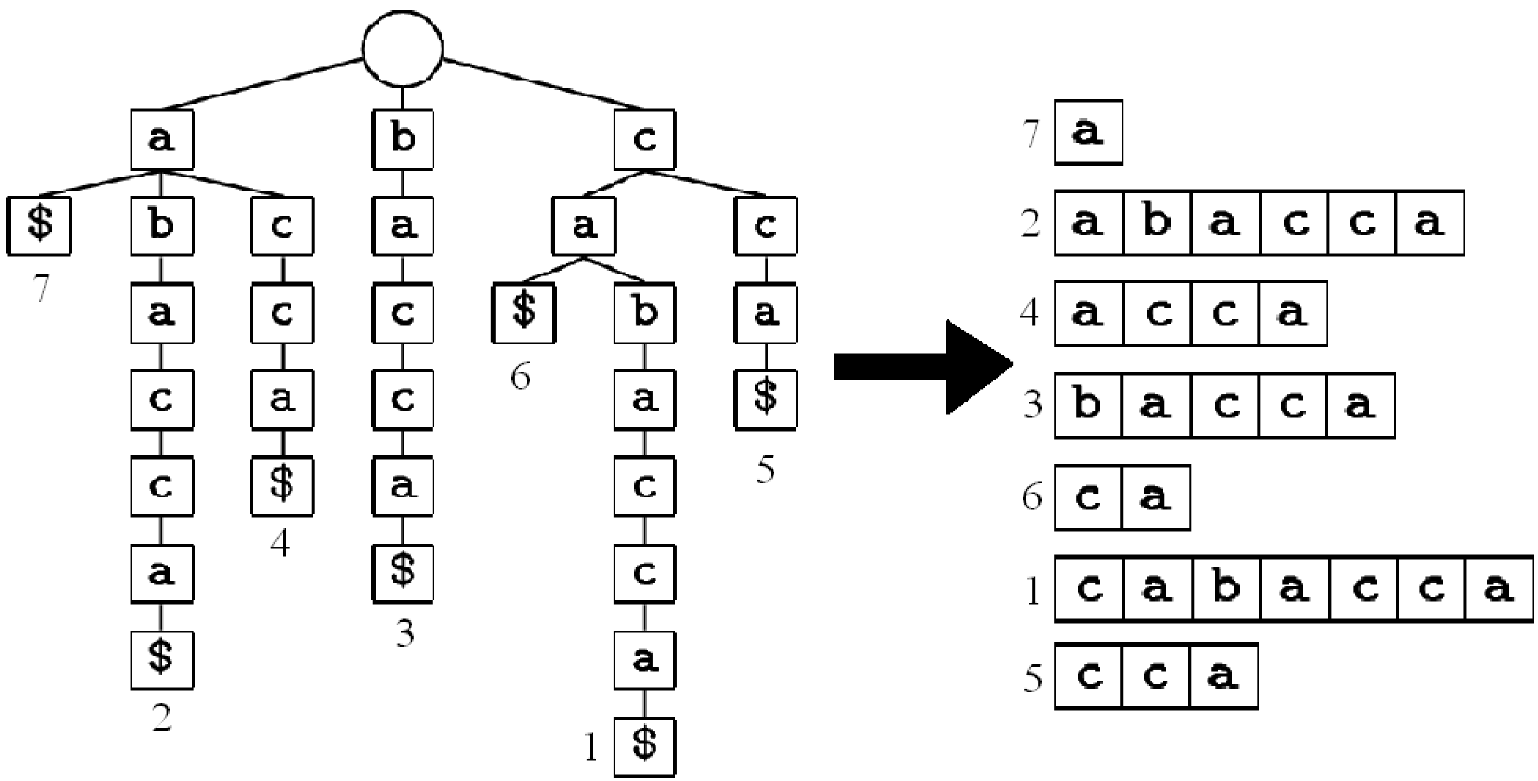
- Преимущества суффиксного массива:
  - Требует мало памяти. Суффиксный массив – два массива длины O(n), содержащие целые числа.
  - Наличие эффективных алгоритмов построения. По скорости часто опережает суффиксные деревья и автоматы.
  - Независимость работы от размера алфавита.
- Недостатки суффиксного массива:
  - Алгоритм построения, работающий за линейное время, сложен и не эффективен.
  - Поиск образца происходит за O(m + log n)

Способы построения

- Существует несколько способов построить массив номеров суффиксов:
  - из суффиксного дерева (O(n))
  - алгоритм Кярккяйнена-Сандерса (O(n))
  - алгоритм построения за O(n log n)
- Далее, зная строку и этот массив, можно с помощью алгоритма Касаи найти LCP соседних суффиксов за O(n).

Из дерева

- Обойти вершины дерева обходом в глубину, посещая детей в лексикографическом порядке. Полученный в результате обхода порядок суффиксов и будет являться суффиксным массивом.



Алгоритм Кярккяйнена-Сандерса

- Придуман в 2006 году.
- Работает за O(n).
- Данный алгоритм достаточно сложен, поэтому его описания не будет в этой работе.

Алгоритм построения за O(n log n)

- Использует идею поразрядной сортировки.
- Алгоритм строит суффиксный массив для зацикленной строки. По этой причине необходимо добавить в конец строки символ, который заведомо меньше всех остальных ($).
- Состоит из нескольких этапов. Перед каждым этапом суффиксы будут отсортированы по первым L буквам, а после по первым 2L буквам. Когда L станет не меньше длины строки, алгоритм прекращает свою работу.
- Перед первым этапом необходимо отсортировать суффиксы по первой букве. Это можно сделать с помощью любой сортировки за O(n log n).

Алгоритм построения за O(n log n) часть 2

- На каждом этапе хранится текущая перестановка суффиксов (в конце это суффиксный массив), обратная перестановка и массив цветов суффиксов.
- Суффиксы, у которых совпадают префиксы длины L, имеют одинаковый цвет. Большие суффиксы имеют больший номер цвета.

Алгоритм построения за O(n log n) часть 3

- Следует заметить, что предыдущие L букв суффиксов – тоже префиксы каких-то суффиксов, а значит мы уже значит их цвета. Поэтому можно считать, что суффиксы отсортированы по вторым L буквам.
- Далее, как в поразрядной сортировке, разложим наши суффиксы по корзинам, в соответствии с цветом первых L букв. Так как мы перебираем суффиксы в порядке цветов вторых L букв, то получаем перестановку суффиксов, отсортированных по первым 2L буквам.

Алгоритм построения за O(n log n) часть 4

- Пример одной фазы:
  - Сдвигаем суффиксы так, чтобы они оказались упорядочены по вторым L буквам. Слева обозначены цвета.

1	$	a	b	a	a	c
2	a	b	a	a	c	$
2	a	a	c	$	a	b
2	a	c	$	a	b	a
3	b	a	a	c	$	a
4	c	$	a	b	a	a

→

c	$	a	b	a	a
$	a	b	a	a	c
b	a	a	c	$	a
a	a	c	$	a	b
a	b	a	a	c	$
a	c	$	a	b	a

Алгоритм построения за O (n log n) часть 5

- Пример одной фазы:
  - Раскладываем суффиксы по новым корзинам. Можно заметить, что корзины будут иметь те же размеры (по цветам первых L букв, изображены слева), но затем их необходимо разбить (по цветам следующих L букв, изображены справа)

c	$	a	b	a	a
$	a	b	a	a	c
b	a	a	c	$	a
a	a	c	$	a	b
a	b	a	a	c	$
a	c	$	a	b	a

→

1	$	a	b	a	a	c	1
2	a	a	c	$	a	b	2
2	a	b	a	a	c	$	3
2	a	c	$	a	b	a	4
3	b	a	a	c	$	a	5
4	c	$	a	b	a	a	6

Алгоритм Касаи

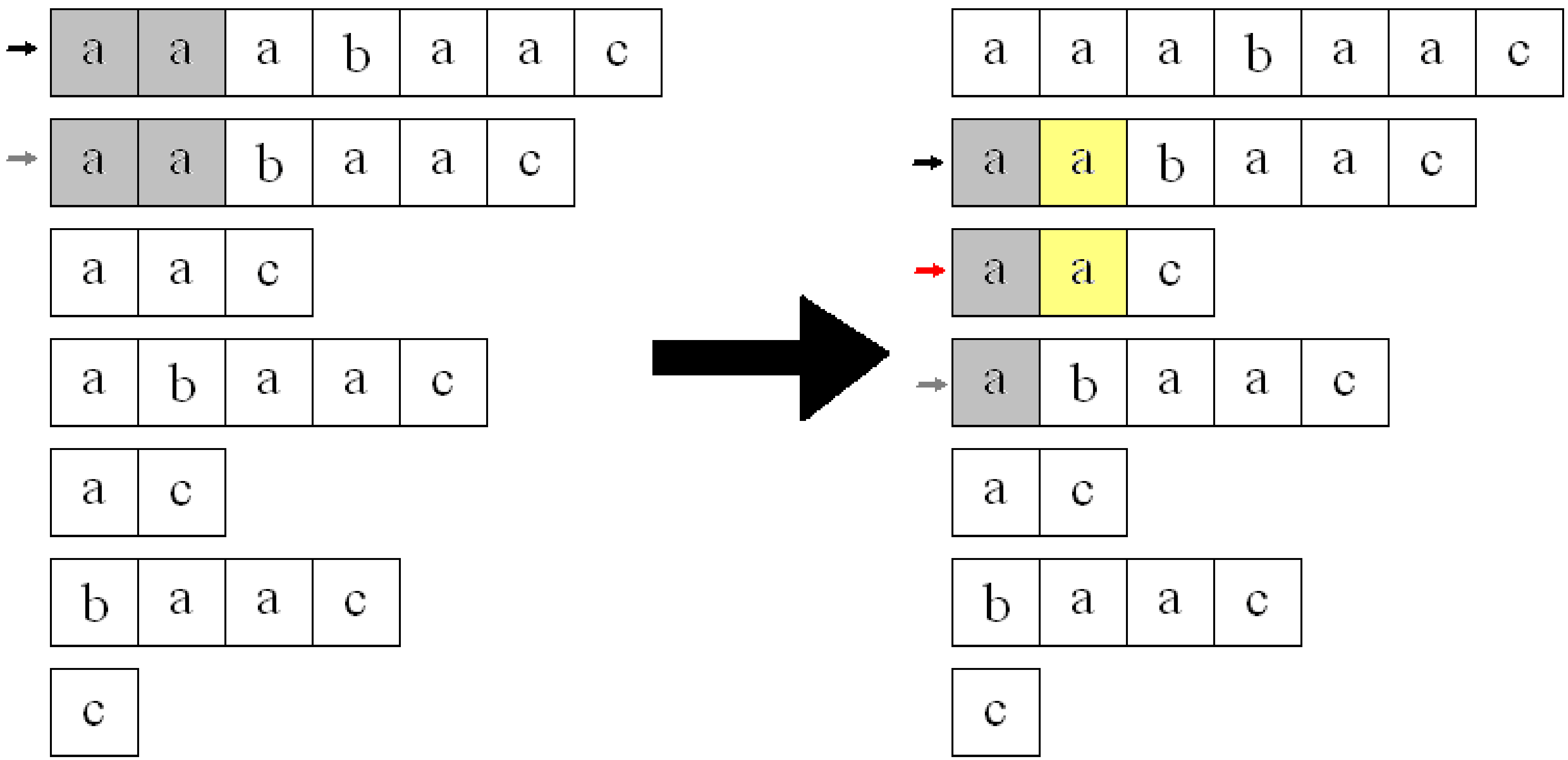
- Позволяет зная строку и массив суффиксов в лексикографическом порядке найти LCP соседних суффиксов.
- Данный алгоритм работает для зацикленной строки.
- Работает за O(n)

Алгоритм Касаи часть 2

- Обычным образом посчитать LCP первых двух суффиксов.
- Заметить, что если убрать у первого из суффиксов по первую букву, то LCP его и следующего за ним не меньше, чем текущий LCP без единицы.
- Обычным образом увеличивать длину общего префикса, пока не будет достигнуто значение LCP .
- Если далее продолжать работу алгоритма тем же методом, то будут найдены LCP всех соседних суффиксов.

Алгоритм Касаи часть 3

- Пример одного шага:



Алгоритм Касаи часть 4

- Данный алгоритм работает за O(n), так как значение LCP не может превышать n и на каждом из n шагов оно уменьшается ровно на 1.
- Далее, имея массив со значениями LCP соседних суффиксов, необходимо построить (за линейное время) на этом массиве дерево отрезков, причем таким образом, чтобы оно было согласовано с границами двоичного поиска.

Поиск образца в суффиксном массиве

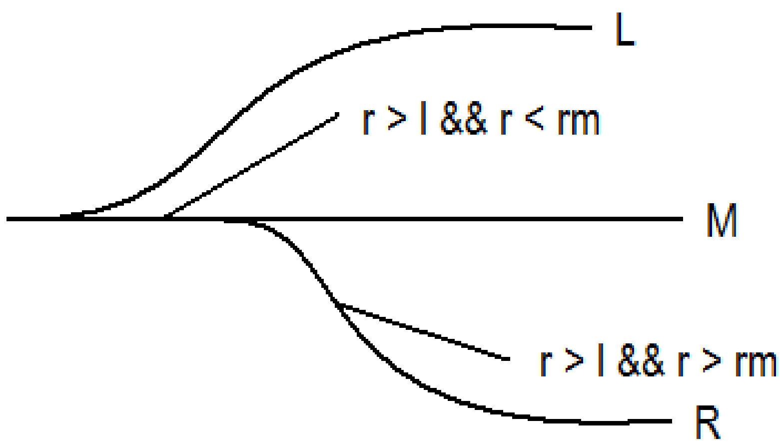
- Простой двоичный поиск
  - Подстрока – префикс некоторого суффикса, поэтому можно использовать двоичный поиск, сравнивая префиксы суффиксов с образцом.
  - Различные вхождения образца будут идти в суффиксном массиве подряд.
  - Время работы за $O(m \log n + k)$, где k – количество вхождений образца в текст.

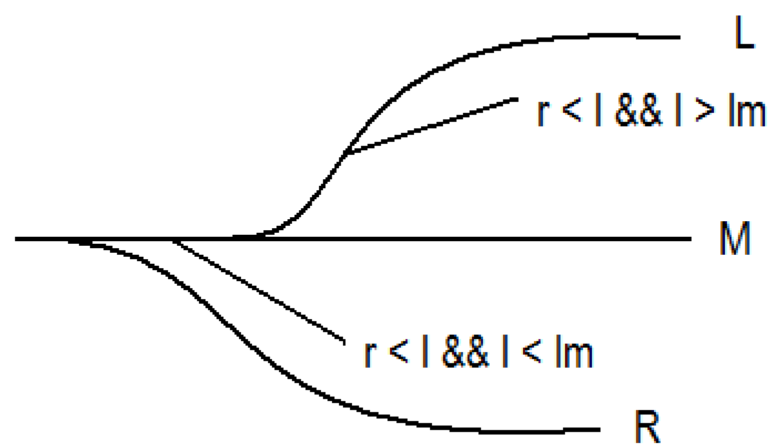
Поиск образца в суффиксном массиве

- Оптимизированный двоичный поиск
  - L – текущая левая граница двоичного поиска
  - R – текущая правая граница двоичного поиска
  - M – текущий рассматриваемый элемент
  - l – LCP образца и левой границы
  - r – LCP образца и правой границы
  - lm – LCP L и M
  - rm – LCP R и M
- l и r перед началом поиска требуется посчитать простым посимвольным сравнением
- lm и rm находятся и RMQ, так как оно согласовано с границами двоичного поиска

Поиск образца в суффиксном массиве

- Исходя из значений r, l, rm и lm можно не сравнивая символы сравнить образец с M.
- В каждом из представленных случаев очевидно, сколько же первых символов образца и M совпадают.





Поиск образца в суффиксном массиве

- Если ни одно из изображенных на рисунке условий не выполняется, то необходимо сравнивать посимвольно, но не с самого начала, а с max(l, r) символа, так как выполняется одно из трех:
  - l = r, тогда хотя бы l первых символов образца и M совпадают
  - l > r и l = lm, тогда первые l символов у L, M и образца совпадают
  - l < r и r = rm , тогда первые r символов у R, M и образца совпадают
- Когда границы поиска сузятся до двух суффиксов, следует проверить эти строки посимвольно.

Поиск образца в суффиксном массиве

- Данный алгоритм работает за O(m + log n)
  - O(log n) — время работы двоичного поиска
  - O(m) — общее время на сравнение символов, так как номер сравниваемого символа в образце не убывает.

Другие использования суффиксного массива

- Поиск количества различных подстрок
- Поиск суммарной длины различных подстрок
- Построение суффиксного дерева из суффиксного массива

Суффиксный автомат

- В зарубежной литературе называется suffix automaton.
- Также называется DAWG (Directed Acyclic Word Graph) – ориентированный ациклический граф слов.

Что это такое?

- Минимальный детерминированный конечный автомат, допускающий только все суффиксы слова, для которого автомат строится.
- Компактная форма хранения суффиксного дерева с объединенными эквивалентными состояниями.

Конечный автомат

По определению, конечный автомат это – M = (Q, q_0, A, E, f), где:

- Q – конечное множество состояний
- q_0 – начальное состояние (принадлежит Q)
- A – конечное множество "допускающих состояний" (принадлежит Q)
- E – конечный входной алфавит
- F – функция переходов, действующая из $Q \times E$ в Q

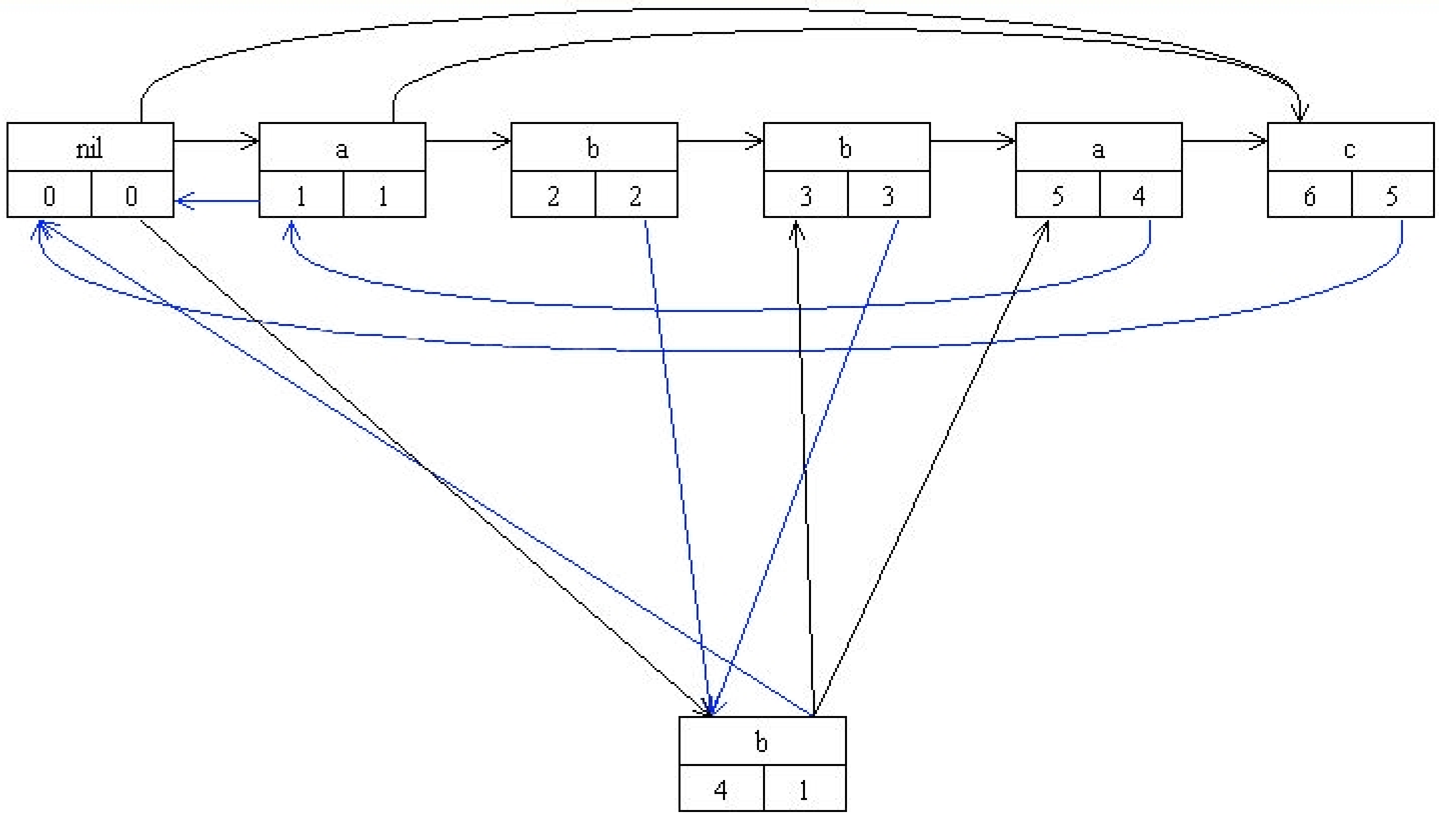
Отличия суффиксного автомата

- Начальное состояние соответствует пустой строке.
- Допускающие состояния соответствуют суффиксам строки.
- В одно состояние идут переходы только по одной и той же букве.
- Допускающими являются все состояния, достижимые по суффиксным ссылкам из состояния, соответствующего всей строке.
- Дополнительная информация – длина наибольшего слова, заканчивающегося в состоянии.

Суффиксные ссылки

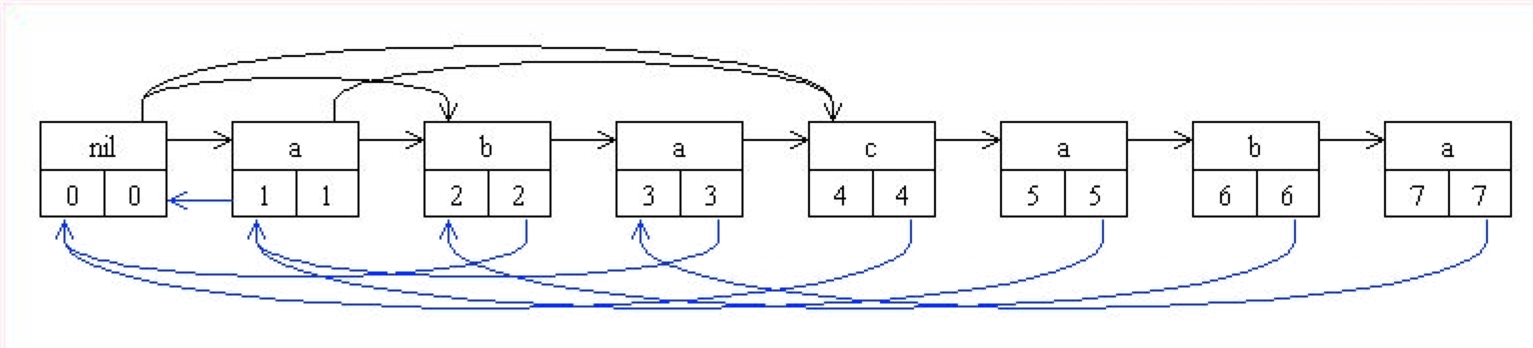
- Суффиксная ссылка – ссылка из вершины, соответствующей слову s, в вершину, соответствующую наидлиннейшему суффиксу s, присутствующую в автомате.
- Суффиксный путь – последовательность вершин, где каждая следующая является вершиной, в которую ведет суффиксная ссылка из предыдущей.

Пример



- строка abbac
- допускающие состояния – №0 и №6

Пример 2



- строка abacaba
- допускающие состояния – №0, №1, №3 и №7

Некоторые факты

- Суффиксный автомат имеет не более 2n-1 состояний.
- Число переходов – не более 3n-4.

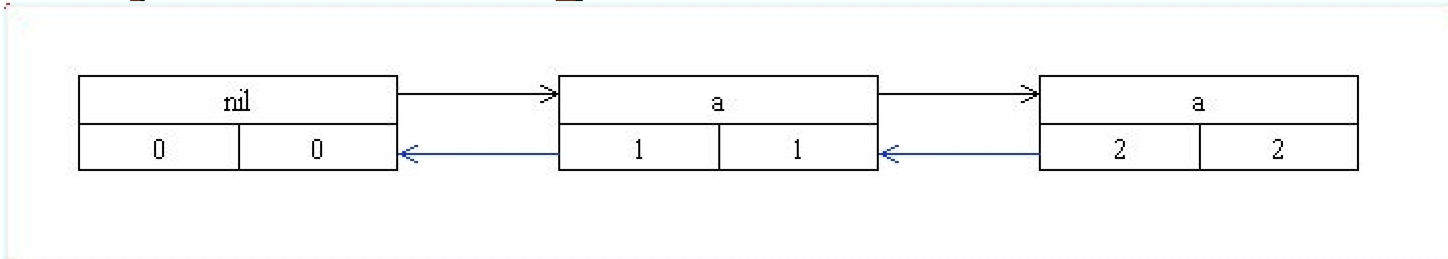
Построение

- За линейное от длины строки время напрямую.
- Из суффиксного дерева за линейное время.
- Из суффиксного дерева для развернутой строки за линейное время.

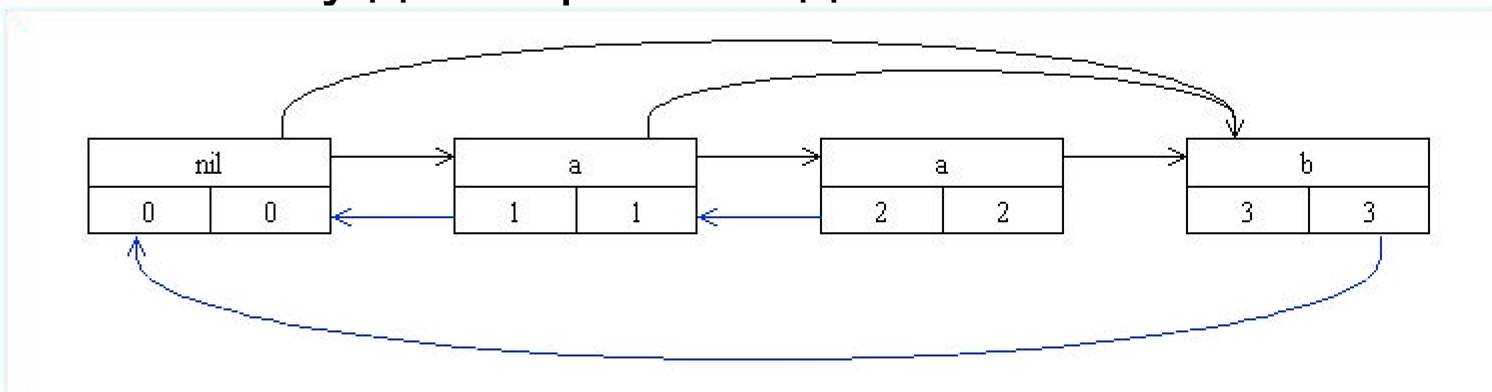
Алгоритм за время O(n)

- На каждом шаге алгоритма к построенному для префикса строки автомату добавляется в конец еще один символ.
- В конец автомата добавляется еще одно состояние.
- Из всех состояний, достижимых из последнего по суффиксным ссылкам (то есть допустимых) добавляются переходы в новое состояние пока не найдется состояние из которого уже есть переход.

Первый случай

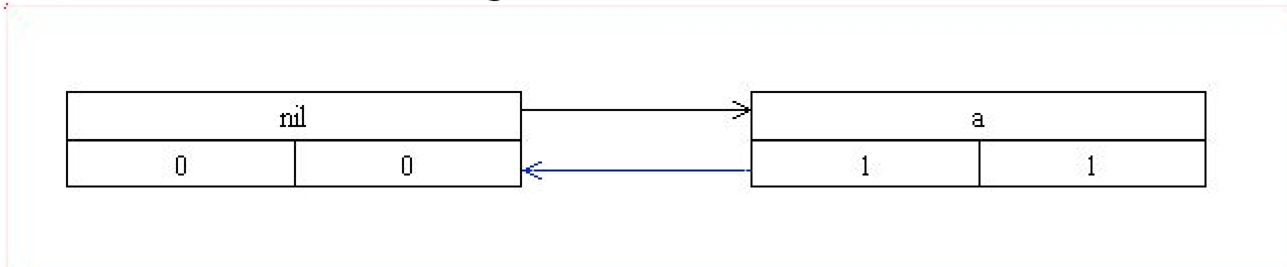


К автомату для строки a a добавляется символ b .

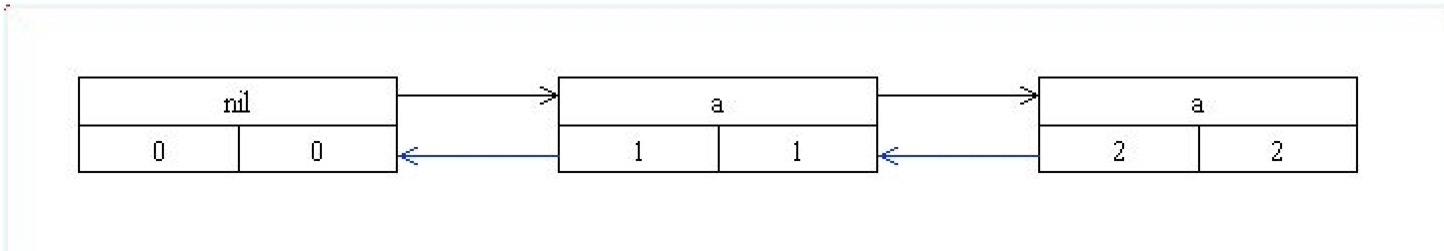


При этом не было найдено состояния, из которого уже есть переход по этому символу, тогда суффиксная ссылка из нового состояния ведет в начальное состояние.

Второй случай

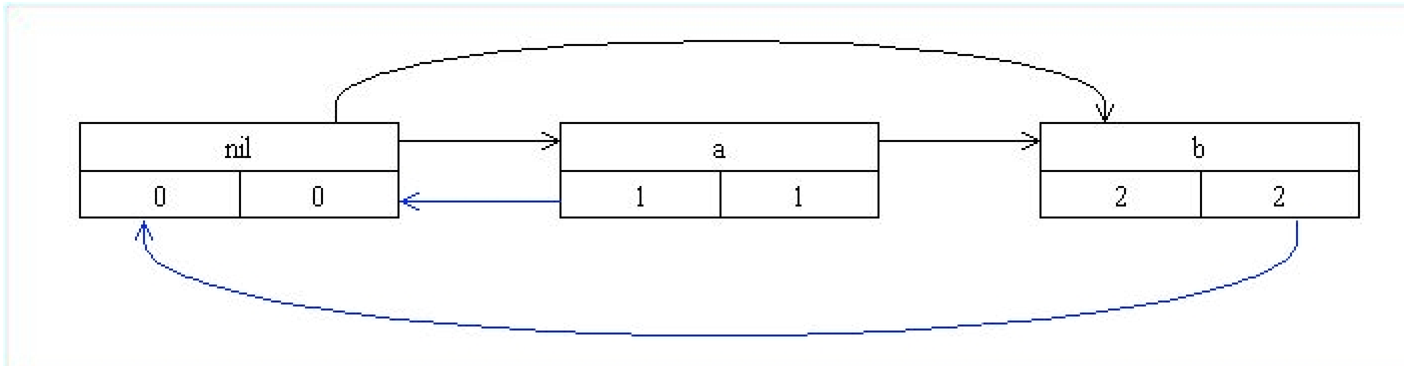


К автомату для строки a добавляется символ a.



Найдено состояние, из которого уже есть переход по символу a, но переход идет в непосредственно следующее состояние (l(1)=l(0)+1). Тогда суффиксная ссылка из нового ведет в это следующее состояние.

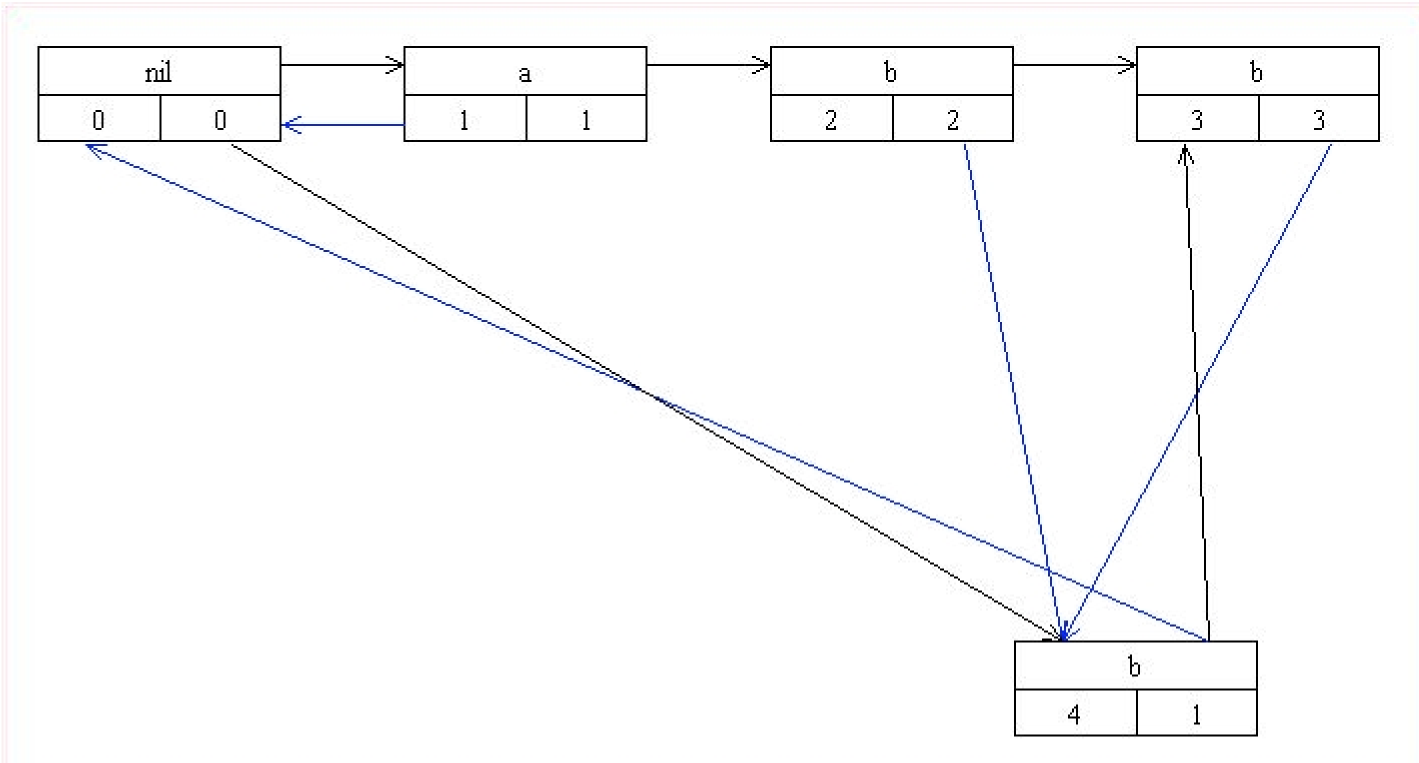
Второй случай



К автомату для строки ab добавляется символ b.

Из состояния №0 есть переход по символу b. Тогда состояние, в которое ведет переход (№2) клонируется, перестраиваются суффиксные ссылки, а все вершины, достижимые из №0 по суффиксным ссылкам, из которых был переход в (№2) перенаправляют этот переход в новый клон.

Результат



Программа

```
void add(char ch) {
        Vertex p = last;
        last = new Vertex();
        last.l = p.l + 1;
        while (p != null && p.next[c] == null) {
                p.next[c] = last;
                p = p.suff;
        }
        if (p == null) {
                last.suff = start;
        } else {
                Vertex q = p.next[c];
                if (q.l <= p.l + 1) {
                        last.suff = q;
                } else {
                        Vertex r = new Vertex();
                        r.l = p.l + 1;
                        last.suff = r; r  .suff = q.suff;   q.suff = r;
                        for (int i = 0; i < q.next.length; i++) {
                                r.next[i] = q.next[i];
                        }
                        while (p != null && p.next[c] == q) {
                                p.next[c] = r;
                                p = p.suff;
                        }
                }
        }
}
```

Применение

Множество задач на строках, например:

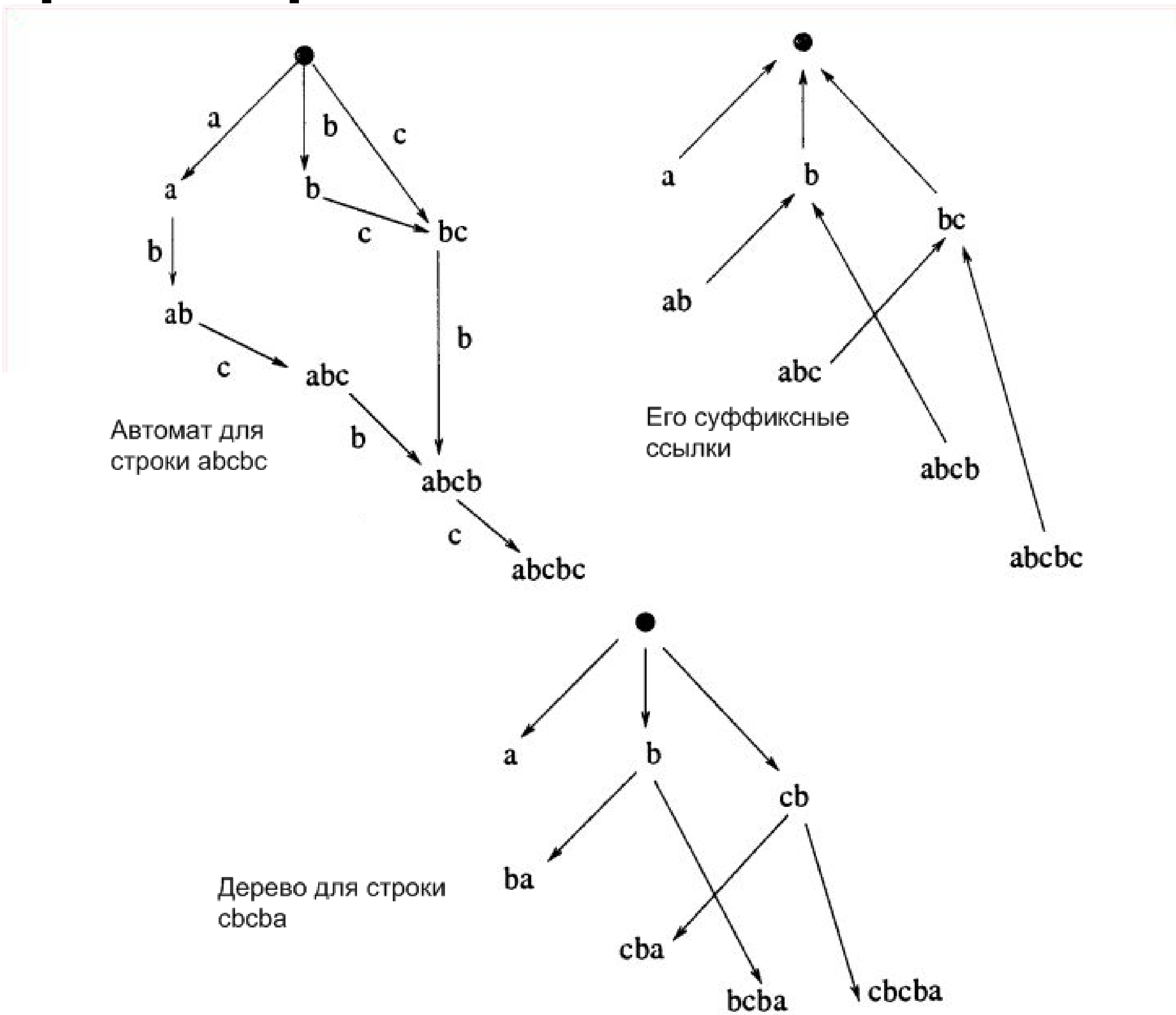
- Проверка того, является ли строка подстрокой другой
- Нахождение наибольшей общей подстроки
- Подсчет количества различных подстрок (с помощью динамического программирования)

Все эти задачи решаются за линейное от длины строки (строк) время (при фиксированном алфавите).

Построение суффиксного дерева из суффиксного автомата для развернутой строки

- Развернутые суффиксные ссылки суффиксного автомата образуют суффиксное дерево для развернутой строки.
- Суффиксные ссылки легко посчитать за линейное время, имея только суффиксное дерево.

Пример


a
b
c
a
b
bc
ab
abc
abcb
abcbc
Автомат для строки abcbc
Его суффиксные ссылки
Дерево для строки cbcba
ba
cb
cba
bcba
cbcba

Источники

- Гасфилд Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. — СПб.: Невский Диалект; БХВ Петербург, 2003.
- Смит Б. Методы и алгоритмы вычислений на строках. — М.: Вильямс, 2006.
- M. Lothaire. Applied Combinatorics on Words.
- M. Crochemore, W. Rytter. Jewels of Stringology.
- M. Crochemore, R. Vérin.
  Direct construction of Compact Directed Acyclic Word Graphs.