Преобразование Берроуза–Вилера
Лекция № 2 курса
«Алгоритмы для Интернета»

Юрий Лифшиц*

5 октября 2006 г.

Содержание

1. Вступление

Во всяком хаосе есть космос,
в каждом беспорядке скрыт тайный порядок.
Карл Юнг

Майкл Берроуз (Michael Burrows) и Дэвид Вилер (David Wheeler) в 1994 году, работая в Digital Equipment Corporation (теперь это Hewlett-Packard), придумали алгоритм, который умеет находить в тексте некоторую регулярность.

Преобразование Берроуза–Вилера (*Burrows–Wheeler transform, BWT*) на входе получает текст и преобразует его в новый текст такой же длины, который легко архивируется благодаря выявленным закономерностям.

На данный момент алгоритм применяется в таких архиваторах, как `bzip2` и `szip`.

*Законспектировал Андрей Законов.

2. Вычисление BWT

2.1. Определение преобразования

Пусть дан текст $S = s_1 \dots s_n$. Добавим в алфавит знак \$, который будем считать последней буквой алфавита. Припишем в конец текста \$ и выпишем все $n + 1$ циклических сдвига. Записывая строчка за строчкой формируемые сдвиги, получим матрицу из $(n + 1) \times (n + 1)$ элементов, где каждая строчка — это один сдвиг. Отсортируем строчки матрицы по алфавиту. Результатом преобразования будет являться последний столбец, назовем его L.

В качестве примера применим преобразование Берроуза–Вилера к тексту $S = racaa$. Дописываем \$ в конец текста, выписываем все циклические сдвиги.

Циклические сдвиги:	Упорядочиваем полученные строки по алфавиту:
$racaa\$$	$aa\$rac$
$acaa\$r$	$acaa\$r$
$caa\$ra$	$a\$raca$
$aa\$rac$	$caa\$ra$
$a\$raca$	$racaa\$$
$\$racaa$	$\$racaa$

Результатом применения преобразования к тексту $S = racaa$ будет столбец $L = (craa\$a)^T$.

Некоторые факты:

- по последнему столбцу можно восстановить текст;
- последний столбец легко сосчитать;
- последний столбец архивируется гораздо лучше, чем исходный текст.

Рассмотрим эти факты подробнее. Для начала ответим на простой вопрос: зная только L, как догадаться, на каком месте в отсортированном списке шел исходный текст S? Достаточно посмотреть на последний столбец. Строка, где стоит \$, является исходным текстом.

На рассмотренном выше примере мы, вычислив матрицу и применив алгоритм сортировки, сосчитали последний столбец. Так ли считают последний столбец в архиваторах? Нет, из-за очевидных недостатков алгоритма: он квадратичный по памяти и требует применения алгоритма сортировки.

2.2. Вычисление в прямую сторону

Рассмотрим алгоритм, работающий за линейное время.

Сначала вспомним такое понятие, как *суффиксное дерево* (рис. 1). Для текста $S = s_1 \dots s_n$ построение суффиксного дерева начнем с того, что припишем специальный символ \$ к тексту. У полученного текста будет $n + 1$ суффикс. Подвесим все суффиксы за начала и склеим все ветки, идущие по одинаковым буквам. В каждом листе записывается номер того суффикса, который в нем заканчивается. *Номером суффикса* будем называть номер позиции в тексте, с которой мы читали суффикс, чтобы прийти в этот лист.

Сделаем небольшую добавку: пусть в нашем суффиксном дереве все ребра сверху вниз идут в алфавитном порядке (ребро по букве a будет самым верхним, по букве b — вторым, и т.д.). Проводим такую сортировку внутри каждой вершины. Алфавит будем считать конечным, значит трудоемкость такой сортировки — фактически константа.

BWT за линейное время: алгоритм

Дописав в конец текста \$, построим суффиксное дерево. Последовательно рассмотрим все внутренние вершины и отсортируем детей каждой из них в алфавитном порядке. Вспомним, что в каждом листе хранится номер конкретного суффикса. Двигаясь сверху вниз, составим последовательность из чисел

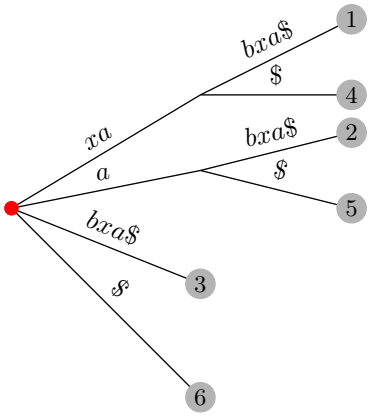


Рис. 1. Пример суффиксного дерева для строки $S = xabxa\$$

во всех листьях. Теперь просматриваем эти числа и для каждого числа i пишем букву s_{i-1} (рис. 5). Полученная последовательность букв будет результатом преобразования.

Сразу возникает логичный вопрос: как обойти все листья за линейное время? Для этого можно применить обход в глубину.

Убедимся, что полученная последовательность и есть последний столбец матрицы сдвигов, в которой строчки отсортированы в алфавитном порядке.

Анализ алгоритма

Для выполнения анализа алгоритма существенными являются следующие наблюдения. Первое важное наблюдение состоит в том, что алфавитный порядок суффиксов совпадает с алфавитным порядком сдвигов. Проиллюстрируем этот факт на примере.

Рассмотрим некоторый текст, допишем \$, сравним два произвольных суффикса: до какого-то момента суффиксы могут быть одинаковыми (рис. 2).

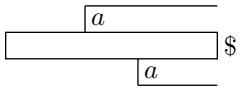


Рис. 2. Начала суффиксов (буква a) совпадают

Найдем место, где появляется расхождение. Рассмотрим отдельно случай, когда это будут различные буквы, например, b и c (рис. 3), и случай, когда вместо одной из букв будет символ \$ (рис. 4).

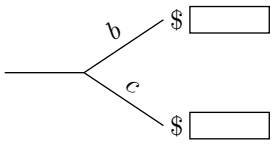


Рис. 3. Суффикс, содержащий b, идет раньше в алфавитном порядке, так как это первое место расхождения, и b раньше по алфавиту, чем c

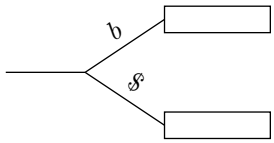


Рис. 4. Вспомним, что \$ ввели как последний символ алфавита. Поэтому, если в первом месте расхождения в одном из суффиксов будет \$, то это значит, что он окажется на нижнем ребре.

Если дописать к суффиксам после символа \$ оставшиеся до циклического сдвига буквы, то алфавитный порядок сохранится.

Рассмотрим еще одно полезное наблюдение. В листьях мы читаем порядок стартов суффиксов. Пусть i — это позиция, с которой начинается соответствующий суффикс, а следовательно, и сдвиг. s_i — буква, с которой начинается этот сдвиг. Тогда s_{i-1} — это последняя буква сдвига. Прочитаем все буквы s_{i-1} в такой очередности, что последовательность букв s_i образует алфавитный порядок. Тогда последовательность последних букв сдвигов s_{i-1} будет искомым последним столбцом L. Проиллюстрируем изложенные утверждения на примере текста $racaa\$$ (рис. 5).

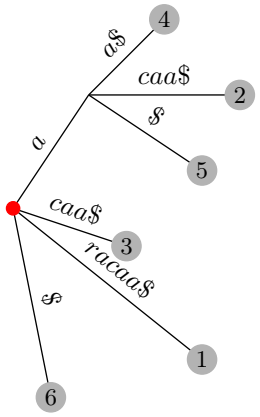


Рис. 5. Вычисление преобразования BWT на примере текста $racaa\$$.

Порядок суффиксов: $aa\$$, $acaa\$$, $a\$$, $caa\$$, $racaa\$$, $\$$.
Порядок сдвигов: $aa\$rac$, $acaa\$r$, $a\$raca$, $caa\$ra$, $racaa\$$, $\$racaa$.
Числа в листьях: $4, 2, 5, 3, 1, 6$ (сверху вниз прочитали все вершины).
Из каждого числа вычитаем единицу и получаем результат из соответствующих букв текста: $s_3, s_1, s_4, s_2, s_6, s_5 = craa\$a$.

Объясним тот факт, что у последнего столбца будет в точности такой же набор букв, как и у исходного текста. Достаточно вспомнить квадратную матрицу, которая содержала все сдвиги. Каждая буква один раз оказывалась в конце.

3. Вычисление обратного преобразования

3.1. Магический вектор T

Определение вектора T

Пусть матрица букв M — это все отсортированные сдвиги текста S. Мы знаем только ее последний столбец L.

Зная последний столбец L, как догадаться до букв первого столбца F? Мы знаем, что L состоит из того же набора букв, что и F. Поэтому достаточно отсортировать столбец L по алфавиту. Вспомогательная матрица M' получается перестановкой столбца L в начало (рис. 6).

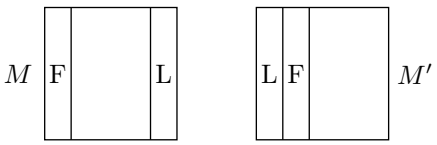


Рис. 6. Матрица M' получена перестановкой последнего столбца L в начало

Матрицы M и M' содержат один и тот же набор строк. Это объясняется тем, что матрица M состоит из всех сдвигов, а матрица M' — из набора всех предыдущих циклических сдвигов. Значит, каждой строчке в M' можно сопоставить строку в M.

Для каждого k определим элемент вектора $T[k]$ так, чтобы k-я строка M' совпадала с $T[k]$-й строкой M. Переформулировка: пусть M_k — это k-й по алфавиту сдвиг текста, тогда предыдущий сдвиг в отсортированном списке сдвигов — это $M_{T[k]}$.

Докажем равносильность этих определений.

Рассмотрим матрицы M и M', а точнее их строчки под номером k (рис. 7). Как из строчек одной матрицы получить строчки другой? Достаточно перенести последнюю букву каждой строки матрицы M в начало, так как по определению матрицы M' ее k-я строка является предыдущим по циклу сдвигом после k-й строки матрицы M.

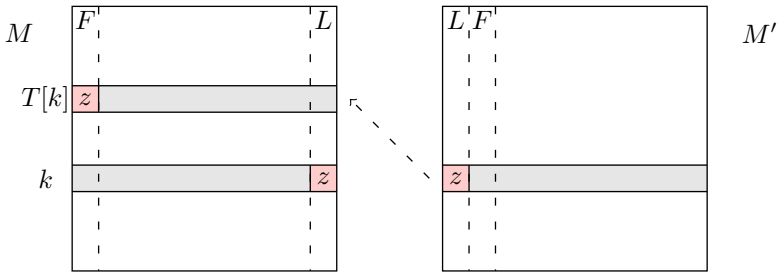


Рис. 7. $T[k]$ — позиция той строки в матрице M, которая идет под номером k в матрице M'

Приведем пример вычисления магического вектора.

$$\begin{pmatrix} a & a & \$ & r & a & c \\ a & c & a & a & \$ & r \\ a & \$ & r & a & c & a \\ c & a & a & \$ & r & a \\ r & a & c & a & a & \$ \\ \$ & r & a & c & a & a \end{pmatrix}$$

Рис. 8. Матрица M

$$\begin{pmatrix} c & a & a & \$ & r & a \\ r & a & c & a & a & \$ \\ a & a & \$ & r & a & c \\ a & c & a & a & \$ & r \\ \$ & r & a & c & a & a \\ a & \$ & r & a & c & a \end{pmatrix}$$

Рис. 9. Матрица M'

Посмотрим на первую строчку $caa\$ra$ матрицы M' (рис. 9). В матрице M (рис. 8) она находится на 4-м месте. Значит, $T[1] = 4$. Смотрим на вторую строчку матрицы M', ищем ее в матрице M, записываем $T[2] = 5$. Продолжая, получим: $T = [4, 5, 1, 2, 6, 3]$.

Все строчки в матрице M' разные, набор строк в матрице M такой же, как и в матрице M'. Фактически вектор T — это перестановка строк матрицы.

Вычисление магического вектора T

Опишем алгоритм, позволяющий за линейное время вычислить вектор T. Первым проходом по буквам столбца L для каждой буквы алфавита α вычислим $C(\alpha)$ — количество раз, которые она встречается в тексте.

Вторым проходом вычислим $D(\alpha)$ — сколько раз встречаются все буквы, идущие до α в алфавитном порядке.

Для каждой позиции i столбца L посчитаем $P(i)$ — сколько раз буква на этой позиции L_i нам уже встречалась выше по столбцу на позициях $j < i$.

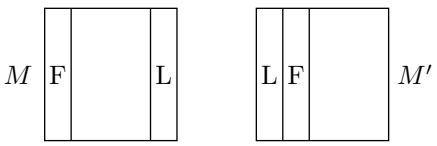


Рис. 10. Рассмотрим две знакомые матрицы: M и M'

Пусть последний столбец $L = (craa\$a)^T$. Прочтем текст и заполним интересующие нас таблицы (рис. 11, 12, 13).

a	c	r	\$
3	1	1	1

Рис. 11. Таблица $C(\alpha)$: столько раз буква α встречается в тексте

a	c	r	\$
0	3	4	5

Рис. 12. Таблица $D(\alpha)$: столько раз встречаются все буквы, идущие в алфавитном порядке до α

c	0
r	0
a	0
a	1
\$	0
a	2

Рис. 13. Таблица $P(i)$: столько раз буква на позиции i нам уже встречалась выше по столбцу

Используя введенные обозначения, получим формулу для T:

$$T[i] = D(L_i) + P(i) + 1.$$

Доказательство. Нам нужно узнать, на каком месте в отсортированном списке находится сдвиг, предыдущий к M_i. Он начинается с буквы L_i — это значит, что он стоит после $D(L_i)$ сдвигов, которые начинаются на меньшие буквы. Среди сдвигов, которые начинаются на L_i, он находится в точности на $P(i) + 1$ месте.

Домашняя задача: как сэкономить один пробег по тексту при подсчете T?

3.2. Восстановление текста с помощью вектора T

Итак, мы знаем последний столбец L и вектор перестановки T. Как узнать последнюю букву текста? Последней буквой всегда будет \$, так как \$ в алфавитном порядке находится на последнем месте и при этом содержится в любом тексте.

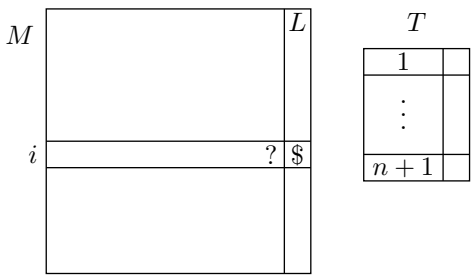


Рис. 14. Как узнать предпоследнюю букву текста?

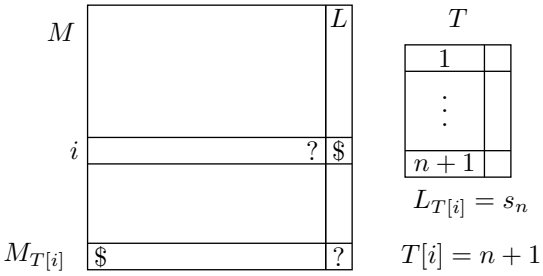


Рис. 15. Предпоследняя буква — это $L_{T[i]}$

Пусть \$ находится на месте i в столбце L (рис. 14). Тогда предпоследняя буква — это $L_{T[i]}$. Действительно, вспомним переформулировку $T[i]$. Как мы знаем, M_i — это исходный текст. Тогда $M_{T[i]}$ — это текст, в котором \$ перенесли в начало. Таким образом, на последнем месте $L_{T[i]}$ оказалась как раз предпоследняя буква (рис. 15).

Перейдем к общему случаю и найдем все буквы текста. Пусть мы уже знаем, что $M_k = s_{j+1} \dots s_n \$ s_1 \dots s_j$, и соответственно, $s_j = L_k$, тогда мы можем узнать предыдущую букву. По определению вектора T мы знаем, что $M_{T[k]} = s_j \dots s_n \$ s_1 \dots s_{j-1}$ (рис. 16).

Получим формулы для всех букв текста от начала к концу.

- Читаем последнюю букву: $\$ = s_{n+1} = L_i$.
- Применяем вектор перестановки T, читаем следующую букву: $s_n = L_{T[i]}$.
- ...

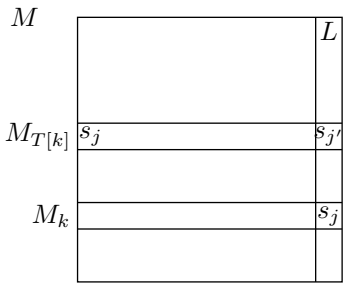


Рис. 16. Получаем $s_{j'} = s_{j-1} = L_{T[k]}$

- Применяем вектор перестановки T, читаем еще одну букву: $s_{n-k} = L_{T^{k+1}[i]}$.
- ...
- n раз применив вектор T, читаем первую букву: $s_1 = L_{T^n[i]}$.

Повторим общий алгоритм обратного BWT.

1. Для каждой буквы алфавита α вычислить количество раз $C(\alpha)$, которые она встречается.
2. Для каждой буквы алфавита α вычислить количество раз $D(\alpha)$, которые встречаются все меньшие ее буквы.
3. Для каждого номера $1 \leqslant i \leqslant n+1$ вычислить количество раз $P(i)$, которое буква L_i уже нам встретилась выше по столбцу.
4. По формуле $T[i] = D(L_i) + P(i) + 1$ вычислить вектор T.
5. По формуле $s_{n-k} = L_{T^{k+1}[i]}$ вычислить все буквы текста от конца к началу.

Рассмотрим пример вычисления обратного преобразования (рис. 17). Пусть *racaa*\$ — исходный текст. После применения прямого преобразования получили столбец $L = craa\$a$ (рис. 5) и вектор $T = [4, 5, 1, 2, 6, 3]$.

Стартовый индекс: $i = 5$;
$s_6 = L_i = L_5 = \$$;
$s_5 = L_{T[5]} = L_6 = a$;
$s_4 = L_{T[6]} = L_3 = a$;
$s_3 = L_{T[3]} = L_1 = c$;
$s_2 = L_{T[1]} = L_4 = a$;
$s_1 = L_{T[4]} = L_2 = r$.

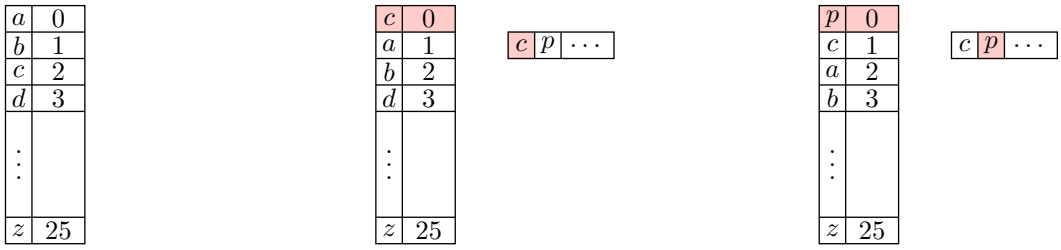
Рис. 17. Демонстрация работы алгоритма, выполняющего обратное преобразование BWT

4. Применение к архивированию

4.1. Кодирование move-to-front

Move-to-front — это перезапись текста в «адаптирующейся» кодировке.

Будем обозначать $Enc(x)$ число, которым кодируется символ x в текущей кодировке. Начинаем с кодировки $Enc(a) = 0, \ldots, Enc(z) = 25$ (рис. 18(а)). Читаем текст слева направо. Прочитав очередную букву α, пишем в выходной поток $Enc(\alpha)$. Сразу же меняем кодировку следующим образом: $Enc(\alpha) = 0$,

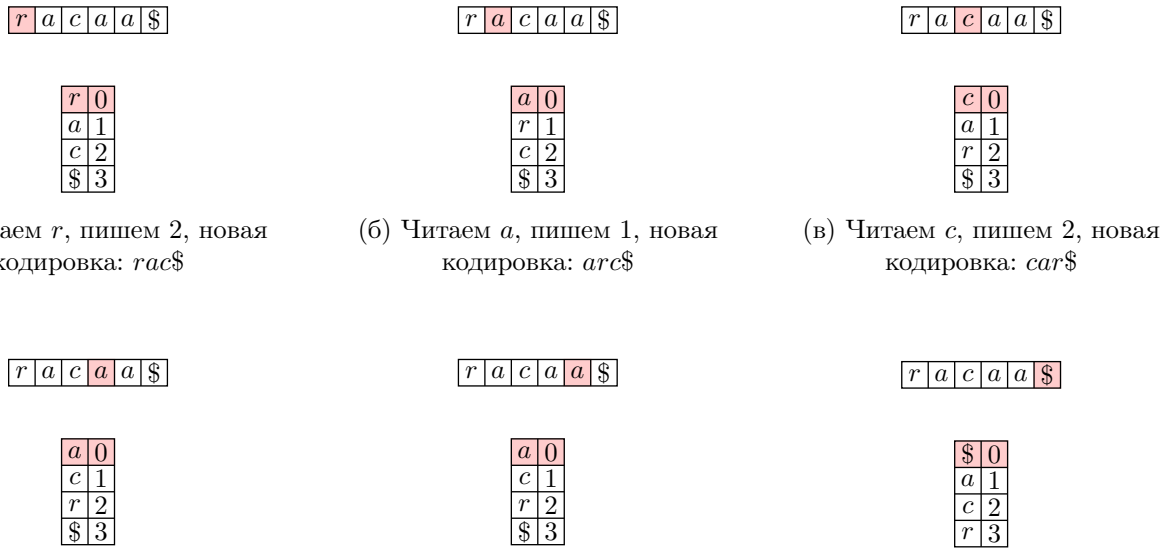


(а) Начальная кодировка (б) Прочитали букву c (в) Прочитали букву p

Рис. 18. Начальная кодировка и примеры изменения кодировки в ходе кодирования

а для всех букв, которые стояли выше α (кодировались меньшими числами), применяем $Enc \leftarrow Enc + 1$ (рис. 18(б–в)). Результат кодирования — последовательность чисел и финальная кодировка.

Проиллюстрируем кодирование на примере текста $racaa\$$ (рис. 19).



(а) Читаем r, пишем 2, новая кодировка: $rac\$$

(б) Читаем a, пишем 1, новая кодировка: $arc\$$

(в) Читаем c, пишем 2, новая кодировка: $car\$$

(г) Читаем a, пишем 1, новая кодировка: $acr\$$

(д) Читаем a, пишем 0, новая кодировка: $acr\$$

(е) Читаем \$, пишем 3, кодировка: $\$acr$

Рис. 19. Кодирование текста $racaa\$$. Начальная кодировка: $Enc(a) = 0$, $Enc(c) = 1$, $Enc(r) = 2$, $Enc(\$) = 3$. Результат: 212103, кодировка: $\$acr$.

4.2. Схема архивирования

1. Применить к тексту прямое преобразование Берроуза–Вилера.
2. Закодировать полученный последний столбец с помощью move-to-front.
3. Полученную числовую последовательность заархивировать классическим архиватором (например, с помощью алгоритма кодирования Хаффмана).

Текст, полученный в результате применения BWT и move-to-front, лучше архивируется, чем исходный, благодаря тому, что после BWT текст становится «локально-однородным» (рис. 20), а после move-to-front кодирования результат содержит много маленьких чисел и мало больших.

Final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $L[i]$ to be the
i	n turn, set $R[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector $R
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with ch appear in the {\em same order
i	n with ch. In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

Рис. 20. Пример отсортированных сдвигов, полученных для текста статьи [1]

Задачи

1. Как по столбцу L вычислить вектор T за два пробега по тексту и один пробег по алфавиту?
2. Можно ли реализовать move-to-front за время $O(n \log |\Sigma|)$, где Σ — используемый алфавит, а n — длина текста?

Источники

[1] M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm
http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf

[2] Mark Nelson. Data Compression with the Burrows-Wheeler Transform
http://marknelson.us/1996/09/01/bwt/

[3] Giovanni Manzini. The Burrows-Wheeler Transform: Theory and Practice
http://www.mfn.unipmn.it/~manzini/papers/mfcs99x.pdf

[4] Юрий Лифшиц. Преобразование Берроуза–Вилера. Аудиолекция
http://rpod.ru/personal/storage/00/00/00/34/23/Yury_Lifshits_Strings_03_BWT.MP3

[5] Страница курса
http://logic.pdmi.ras.ru/~yura/internet.html