Содержание

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Maker Media очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Предисловие

Эта книга посвящена алгоритмам обработки текста, которые иногда называют алгоритмической стрингологией (stringology). Текст (слово, строка, последовательность строк) – один из основных типов неструктурированных данных, играющий важную роль в информатике.

Предмет нашего рассмотрения многогранный, потому что лежит в основе многих дисциплин, особенно информатики и инженерных наук. Исследование неструктурированных данных – активно развивающаяся область, требующая эффективных методов как вследствие присутствия в разных местах операционных систем, так и для анализа огромного объема данных, порождаемых цифровыми сетями и оборудованием. Последнее относится прежде всего к компаниям в сфере ИТ, которые управляют гигантскими массивами данных в ЦОДах, но также ко многим научным направлениям за пределами информатики.

В этой книге представлен репрезентативный набор самых интересных задач в области обработки текстов. Лаконичное и увлекательное изложение открывает путь к более сложным темам. Материалы были взяты из сотен серьезных научных публикаций – каким-то из них уже сотни лет, а какие-то были написаны совсем недавно. По большей части задачи связаны с конкретными приложениями, но есть и более абстрактные. В основе большинства задач лежит остроумный короткий алгоритм, исключение составляют разве что несколько вводных комбинаторных проблем.

Эта книга – не просто очередная монография, а серия задач (головоломок и упражнений). Ее можно рассматривать как дополнение к книгам на эту тему, в которых предмет излагается более полно, в академическом стиле. Тем не менее большинство относящихся к предмету идей включено, так что книга заполняет доселе существовавший пробел и представляет долгожданный подарок для студентов и преподавателей, поскольку является первым задачником с решениями на тему алгоритмов обработки текста.

Книга состоит из семи глав.

- «Первые понятия стрингологии» – вводная глава, в которой читатель знакомится с терминологией, основными понятиями и инструментарием, который будет использоваться в последующих главах и отражает шесть основных направлений в рассматриваемой области.
- Глава «Комбинаторные задачи» посвящена комбинаторным свойствам слов. Это важная тема, поскольку комбинаторные свойства входных данных лежат в основе многих алгоритмов.
- Глава «Сопоставление с образцом» – классическая тема, охватывающая поиск в тексте и сопоставление строк.
- Глава «Эффективные структуры данных» посвящена структурам данных для индексирования текстов, в частности специализированным для текстов массивам и деревьям. Они применяются во многих алгоритмах.

- В главе «Регулярные структуры в словах» рассматриваются регулярные структуры, встречающиеся в текстах, в частности повторения и симметрии, от которых существенно зависит эффективность алгоритмов.
- Глава «Сжатие текста» посвящена методам, которые особенно важны для сжатия текста без потери информации.
- В главе «Разное» рассматриваются задачи, которым не нашлось места в предыдущих главах, но которые, без сомнения, заслуживают внимания.

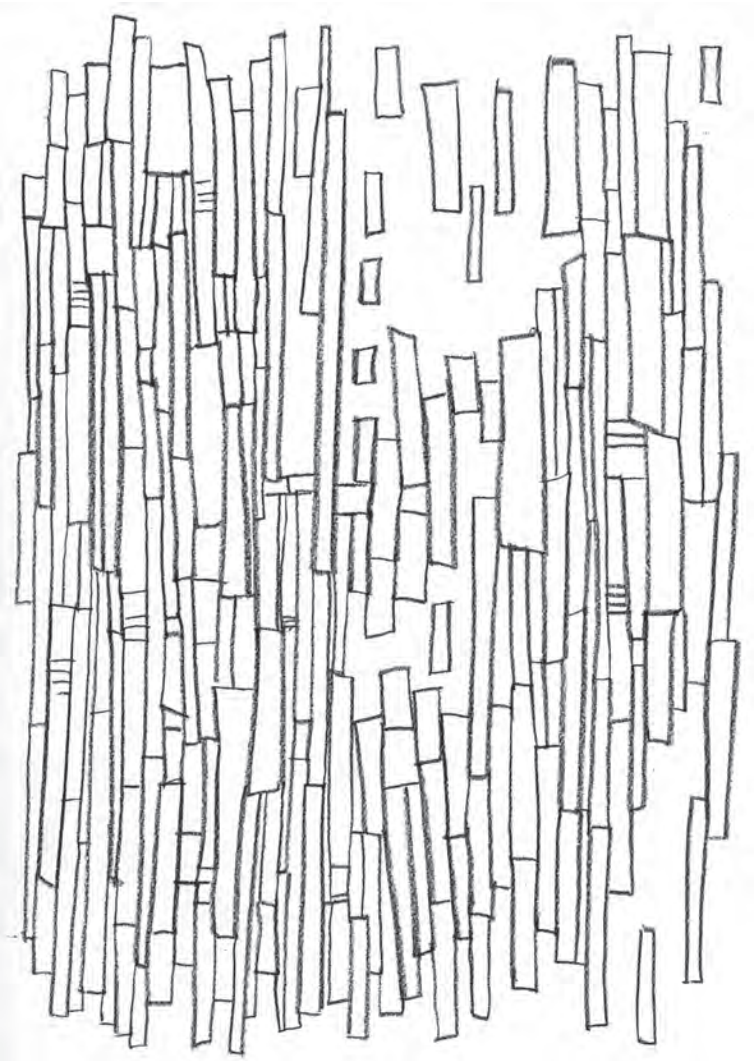
Включенные в книгу задачи отбирались и шлифовались на протяжении нескольких лет преподавания алгоритмов обработки текста в учебных учреждениях Франции, Польши, Великобритании и США, в основном студентам-магистрантам. Они приводятся вместе с решениями и со ссылками для дальнейшего изучения. Учитывался также предыдущий опыт авторов в написании учебников.

Преподаватели курсов по структурам данных и алгоритмам для магистрантов могут взять из книги те темы, которые считают наиболее подходящими для студентов. Однако книга в целом не является элементарной и задумывалась как справочник для исследователей, магистрантов и докторантов, а также для преподавателей вузов, читающих курсы по алгоритмам, в т. ч. не связанным напрямую с обработкой текстов. Ее следует рассматривать как дополнение к стандартным учебникам по данному предмету. Книга содержит всю необходимую для понимания информацию, поэтому позволяет быстро войти в курс дела и разобраться в постановке задач и их решениях без предварительной подготовки.

Издание может быть полезно для специальных курсов по алгоритмам обработки текста и для более общих курсов по алгоритмам и структурам данных. Хотя в нем вводятся все понятия и идеи, необходимые для решения задач, но подготовка в объеме курса по алгоритмам, структурам данных и дискретной математике, рассчитанного на студентов-второкурсников, была бы целесообразна для усвоения материала.

Глава 1

Первые понятия стрингологии



В этой главе мы познакомимся с обозначениями и определениями, а также кратко обрисуем некоторые конструкции, используемые в алгоритмах обработки текста.

Текст – центральный объект в «текстовых процессорах», которые обычно имеют дело с довольно большими объектами. Алгоритмы обработки текстов встречаются в различных областях науки и анализа информации. Во многих текстовых редакторах и языках программирования имеются средства обработки текстов. Например, в молекулярной биологии подобные алгоритмы применяются к анализу молекулярных последовательностей.

Слова

Алфавит – это непустое множество, элементы которого называются ***буквами***, или символами. Обычно используются алфавиты $\mathbf{A} = \{\mathtt{a},\mathtt{b},\mathtt{c},\ldots\}$, $\mathbf{B} = \{\mathtt{0},\mathtt{1}\}$ и натуральные числа. ***Словом***, или ***строкой***, над алфавитом A называется последовательность элементов A.

Последовательность букв нулевой длины называется ***пустым словом*** и обозначается ε. Множество всех конечных слов над алфавитом A обозначается A^*, а $A^+ = A^* \setminus \{\varepsilon\}$.

Длина слова x, т. е. длина последовательности букв, обозначается $|x|$. Буква в позиции $i = 0, 1, \ldots, |x| - 1$ непустого слова x обозначается $x[i]$, а i иногда называют ***индексом*** буквы. Последовательность $x = x[0]x[1]\cdots x[|x| - 1]$ обозначают также $x[0..|x| - 1]$. Множество букв, встречающихся в слове x, обозначается $alph(x)$. Например, для $x = \mathtt{abaaab}$ имеем $|x| = 6$ и $alph(x) = \{\mathtt{a},\mathtt{b}\}$.

Произведением, или ***конкатенацией***, двух слов x и y называется слово, образованное буквами x, за которыми следуют буквы y. Оно обозначается xy или $x \cdot y$, чтобы показать, из каких слов составлен результат. Нейтральным элементом относительно этой операции является пустое слово ε, а zy^{-1} и $x^{-1}z$ соответственно обозначаются такие слова x и y, для которых $z = xy$.

Сопряженным к слову x, а также ***ротацией*** или ***циклическим сдвигом*** x называется любое слово y, которое можно представить в виде vu, где $uv = x$. Это определение имеет смысл, потому что произведение слов, очевидно, некоммутативно. Например, множество слов, сопряженных к $\mathtt{abba}$, – его класс сопряженности (потому что сопряженность – отношение эквивалентности) – равно $\{\mathtt{aabb},\mathtt{abba},\mathtt{baab},\mathtt{bbaa}\}$, а класс сопряженности $\mathtt{abab}$ равен $\{\mathtt{abab},\mathtt{baba}\}$.

Слово x называется ***фактором*** (или ***подстрокой***) слова y, если существуют такие слова u и v, что $y = uxv$. Если $u = \varepsilon$, то говорят, что x является ***префиксом*** y, а если $v = \varepsilon$, то x является ***суффиксом*** y. Множества факторов, префиксов и суффиксов слова x обозначаются соответственно $Fact(x)$, $Pref(x)$ и $Suff(x)$.

Если x – непустой фактор $y = y[0..n-1]$, то он имеет вид $y[i..i + |x| - 1]$ для некоторого i. ***Вхождением*** x в y называется интервал $[i..i + |x| - 1]$, для которого $x = y[i..i + |x| - 1]$. Говорят, что i – ***начальная*** (или левая) ***позиция*** этого вхождения, а $i + |x| - 1$ – ***конечная*** (или правая) ***позиция***. Вхождение x в y можно также определить как тройку (u, x, v) такую, что $y = uxv$. Тогда начальная позиция вхождения равна $|u|$. Например, начальные и конечные позиции слова $x =$ aba в слове $y =$ babaababa равны

i	0	1	2	3	4	5	6	7	8
$y[i]$	b	a	b	a	a	b	a	b	a
Начальные позиции		1			4		6		
Конечные позиции				3			6		8

Для слов x и y $|y|_x$ обозначает количество вхождений x в y. В частности, $|y| = \sum\{|y|_a : a \in alph(y)\}$.

Слово x называется ***подпоследовательностью***, или ***подсловом*** y, если y разлагается в произведение $w_0x[0]w_1x[1]...x[|x| - 1]w_{|x|}$ для некоторых слов w_0, $w_1, ..., w_{|x|}$.

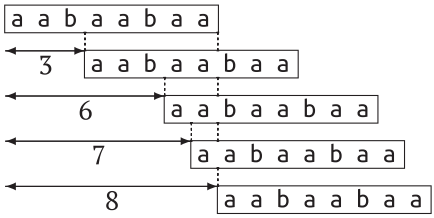
Говорят, что фактор, или подпоследовательность, x слова y является ***собственным***, если $x \neq y$.

Периодичность

Пусть x – непустое слово. Целое число p, $0 < p \leq |x|$, называется ***периодом*** x, если $x[i] = x[i + p]$ для $i = 0, 1, ..., |x| - p - 1$. Заметим, что длина слова является его периодом, так что любое непустое слово имеется хотя бы один период. ***Наименьший период*** x обозначается $per(x)$. Например, 3, 6, 7 и 8 являются периодами слова aabaabaa, а $per($aabaabaa$) = 3$. Отметим, что если p – период x, то все его кратные, не превосходящие $|x|$, также являются периодами x.

Перечислим свойства, эквивалентные определению периода p слова x. Во-первых, x можно единственным способом разложить в произведение $(uv)^k u$, где u и v – слова, причем v не пусто, k – положительное целое число и $p = |uv|$. Во-вторых, x является префиксом ux для некоторого слова u длины p. В-третьих, x является фактором u^k, где u – слово длины p, а k – целое положительное число. В-четвертых, x можно представить в виде произведения $uw = wv$, где u, v и w – слова и $p = |u| = |v|$.

И напоследок введем понятие границы. ***Границей*** слова x называется собственный фактор x, являющийся одновременно префиксом и суффиксом x. Граница x наибольшей длины обозначается $Border(x)$. Таким образом, ε, a, aa и aabaa – границы aabaabaa, а $Border($aabaabaa$) =$ aabaa.

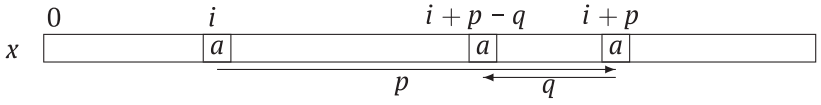


Между границами и периодами x имеется взаимно однозначное соответствие в силу четвертого из приведенных выше утверждений: с периодом p слова x ассоциирована граница $x[p..|x|-1]$.

Заметим, что, в силу определения, граница границы x также является границей x. А раз так, то список $\langle Border(x), Border^2(x), \ldots, Border^k(x) = \varepsilon\rangle$ содержит все границы x. Говорят, что непустое слово x ***свободно от границ***, если его единственной границей является пустое слово, или, что эквивалентно, если его единственным периодом является $|x|$.

Лемма* 1 *(лемма о периодичности). *Если p и q – периоды слова x и $p + q - \gcd(p, q) \le |x|$, то $\gcd(p, q)$ также является периодом x*[1].

Доказательство можно найти в учебниках (см. примечания). Слабой леммой о периодичности называется вариант этой леммы с более сильным условием: $p + q \le |x|$. Ее доказательство не вызывает затруднений и приведено ниже:

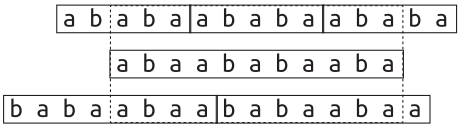


Утверждение леммы, очевидно, справедливо, если $p = q$. В противном случае без ограничения общности предположим, что $p > q$, и покажем сначала, что $p - q$ – период x. Обозначим i позицию в x, для которой $i + p < |x|$. Тогда $x[i] = x[i + p] = x[i + p - q]$, потому что p и q – периоды. И если $i + p \ge |x|$, то из условия следует, что $i - q \ge 0$. Тогда $x[i] = x[i - q] = x[i + p - q]$, как и прежде. Таким образом, $p - q$ – период x. Повторяя это рассуждение или воспользовавшись рекурсией, как в алгоритме Евклида, заключаем, что $\gcd(p, q)$ – период x.

Для иллюстрации леммы о периодичности рассмотрим слово x, допускающее периоды 5 и 8. Если дополнительно предположить, что x включает хотя бы две разные буквы, то $\gcd(5, 8) = 1$ не является периодом x. Таким образом, условие леммы не может удовлетворяться, а значит, $|x| < 5 + 8 - \gcd(5, 8) = 12$.

---

[1] gcd (greatest common divisor) – наибольший общий делитель. – *Прим. перев.*



Пример, изображенный на рисунке, показывает, что условие на периоды в лемме о периодичности не может быть ослаблено.

Регулярные структуры

Степень слова x определяется следующим образом: $x^0 = \varepsilon$, $x^i = x^{i-1}x$ для целого положительного i. k-я ***степень*** x обозначается x^k. Она называется ***квадратом***, если k – положительное четное число, и ***кубом***, если k – положительное, кратное 3.

Следующая лемма является первым следствием из леммы о периодичности.

Лемма 2. *Для слов x и y равенство $xy = yx$ имеет место тогда и только тогда, когда x и y – целые степени одного и того же слова. То же утверждение справедливо, когда существует два целых положительных числа k и l таких, что $x^k = y^l$.*

Доказательства обеих частей леммы по существу одинаковы (на самом деле лемма вытекает из более общего утверждения о кодах). Например, если $xy = yx$, то x и y являются границами слова, следовательно, $|x|$ и $|y|$ – его периоды, а в силу леммы о периодичности периодом является также $\gcd(|x|, |y|)$. Из того, что $\gcd(|x|, |y|)$ делит также $|xy|$, получаем требуемое утверждение. Обратное утверждение очевидно.

Непустое слово x называется ***примитивным***, если оно не является степенью никакого другого слова. Иными словами, x примитивное, если из того, что $x = u^k$ для некоторого слова u и целого положительного k, следует, что $k = 1$, а значит, $u = x$. Например, слово `abaab` примитивное, тогда как ε и `bababa` $=$ $(\mathtt{ba})^3$ таковыми не являются.

Из леммы 2 следует, что всякое непустое слово является степенью ровно одного примитивного слова. Если $x = u^k$ и u примитивное, то u называется ***примитивным корнем*** из x, а k – его ***показателем степени***, обозначаемым $\exp(x)$. Вообще, показателем степени x называется величина $\exp(x) = |x|/per(x)$, необязательно целая, а слово называется ***периодическим***, если его показатель степени не меньше 2.

Отметим, что количество слов, сопряженных данному, т. е. размер его ***класса сопряженности***, равно длине примитивного корня из него.

Следующее утверждение также вытекает из леммы о периодичности.

Лемма 3 (лемма о примитивности, лемма о синхронизации). *Непустое слово x примитивное тогда и только тогда, когда оно входит фактором в свой квадрат только в виде префикса и суффикса, или, эквивалентно, тогда и только тогда, когда $per(x^2) = |x|$.*

Результат этой леммы показан на рисунке. Слово abbaba примитивное, поэтому входит в свой квадрат только двумя способами, тогда как слово ababab примитивным не является и в свой квадрат входит четырьмя способами.

Понятие ***серии***, или ***максимальной периодичности***, охватывает несколько типов регулярных структур в словах. Серией (run) в слове x называется максимальное вхождение периодического фактора. Более формально, это интервал $[i..j]$ позиций внутри x, для которого $\exp(x[i..j]) \geq 2$ и периоды $x[i-1..j]$ и $x[i..j+1]$, если они существуют, больше, чем период $[i..j]$. В этой ситуации, поскольку вхождение идентифицируется индексами i и j, мы также, допуская вольность речи, будем говорить, что $x[i..j]$ – серия.

Еще одна регулярность – наличие в словах взаимно обратных факторов, или палиндромов. ***Обращением***, или ***зеркальным отражением***, слова x называется слово $x^R = x[|x|-1]x[|x|-2]...x[0]$. С этой операцией связано понятие ***палиндрома***: слова, для которого $x^R = x$.

Например, в английском языке слова noon и testset являются палиндромами. Первое – четный палиндром вида uu^R, а второе – нечетный палиндром вида uau^R, где a – буква. Букву a можно заменить коротким словом, что ведет к понятию разрывного палиндрома, полезному при рассмотрении операций фолдинга, встречающихся в последовательностях биологических молекул. Еще пример: целые числа, десятичное представление которых является палиндромом, кратны 11; так, $1661 = 11 \times 151$, $175\,571 = 11 \times 15\,961$.

Упорядочение

Для некоторых алгоритмов важно существование отношения порядка в алфавите, обозначаемого символом $\leq$. Это отношение порядка следующим образом индуцирует ***лексикографический***, или ***алфавитный***, ***порядок*** на множестве слов. Как и порядок в самом алфавите, оно обозначается символом $\leq$. Для $x, y \in A^*$ $x \leq y$ тогда и только тогда, когда либо x является префиксом y, либо x и y можно представить в виде $x = uav$ и $y = ubw$, где u, v и w – слова, а a и b – буквы, причем $a < b$. Так, ababb < abba < abbaab, если считать, что a < b, и вообще на алфавите **A** определен естественный порядок.

Мы говорим, что x ***строго меньше*** y, и обозначаем это $x \ll y$, если $x \leq y$, но x не является префиксом y. Отметим, что из $x \ll y$ следует, что $xu \ll yv$ для любых слов u и v.

На базе лексикографического порядка определяются понятия ***слов Линдона*** и ***ожерелий***.

Словом Линдона x называется примитивное слово, наименьшее среди всех своих сопряженных. Эквивалентно, хотя это и не очевидно, можно сказать, что x меньше любого из своих собственных непустых суффиксов, поэтому его также называют ***самоминимальным словом***. Как следствие x не име-

ет границы. Известно, что любое непустое слово w допускает единственное разложение в произведение $x_0x_1...x_k$, где все x_i – слова Линдона и $x_0 \geq x_1 \geq ... \geq x_k$. Например, слово aababaabaaba раскладывается в произведение aabab · aab · aab · a, где aabab, aab и a – слова Линдона.

Ожерельем, или ***минимальным словом***, называется слово, наименьшее в своем классе сопряженности. Оно является (целой) степенью слова Линдона. Слово Линдона является ожерельем, но обратное неверно; например, $\mathtt{aabaab} = \mathtt{aab}^2$ – ожерелье, но не слово Линдона.

Примечательные слова

Помимо слов Линдона, есть еще три класса слов, которые обладают примечательными свойствами и часто используются в примерах. Это слова Туэ–Морса, слова Фибоначчи и слова де Брёйна. Первые два – префиксы (односторонне) бесконечных слов. Формально ***бесконечным словом*** над алфавитом A называется отображение множества натуральных чисел в A. Множество бесконечных слов обозначается A^∞.

Понятие (моноидного) ***морфизма*** является центральным для определения некоторых бесконечных множеств слов или ассоциированных с ними бесконечных слов. Морфизмом из A^* в себя (или в другой свободный моноид) называется отображение $h : A^* \to A^*$ такое, что $h(uv) = h(u)h(v)$ для любых слов u и v. Следовательно, морфизм полностью определяется образами $h(a)$ букв $a \in A$.

Слово Туэ–Морса порождается повторным применением ***морфизма Туэ–Морса*** μ из $\{\mathtt{a},\mathtt{b}\}^*$ в себя, определенного следующим образом:

$$\begin{cases}\mu(\mathtt{a}) = \mathtt{ab},\\ \mu(\mathtt{b}) = \mathtt{ba}.\end{cases}$$

Итеративное применение этого морфизма, начиная с буквы a, дает список ***слов Туэ–Морса*** $\mu^k(\mathtt{a})$, $k \geq 0$. Первые несколько таких слов показаны ниже:

$\tau_0 = \mu^0(\mathtt{a}) = \mathtt{a}$
$\tau_1 = \mu^1(\mathtt{a}) = \mathtt{ab}$
$\tau_2 = \mu^2(\mathtt{a}) = \mathtt{abba}$
$\tau_3 = \mu^3(\mathtt{a}) = \mathtt{abbabaab}$
$\tau_4 = \mu^4(\mathtt{a}) = \mathtt{abbabaabbaababba}$
$\tau_5 = \mu^5(\mathtt{a}) = \mathtt{abbabaabbaababbabaababbaabbabaab}$

В пределе получается такое ассоциированное бесконечное слово:

$$\mathbf{t} = \lim_{k\to\infty} \mu^k(\mathtt{a}) = \mathtt{abbabaabbaababbabaababbaabbabaab}\ldots.$$

Эквивалентное определение слов Туэ–Морса дает следующее рекуррентное соотношение:

$$\begin{cases}\tau_0 = \mathtt{a},\\ \tau_{k+1} = \tau_k\overline{\tau_k}, \quad k \geq 0,\end{cases}$$

где знак надчеркивания обозначает морфизм $\overline{\mathtt{a}} = \mathtt{b}$, $\overline{\mathtt{b}} = \mathtt{a}$. Отметим, что длина k-го слова Туэ–Морса $|\tau_k| = 2^k$.

Прямое определение **t** выглядит так: буква **t**[n] равна b, если число единиц в двоичном представлении n нечетно, и равна a в противном случае.

Известно, что бесконечное слово Туэ–Морса не содержит перекрытий (факторов вида *аиаиа*, где *а* – буква, а *и* – слово), т. е. факторов с показателем степени больше 2. Говорят, что оно ***свободно от перекрытий***.

Слово Фибоначчи тоже порождается повторным применением морфизма, а именно *морфизма Фибоначчи* φ, отображающего $\{\mathtt{a},\mathtt{b}\}^*$ в себя и определенного следующим образом:

$$\begin{cases}\varphi(\mathtt{a}) = \mathtt{ab},\\ \varphi(\mathtt{b}) = \mathtt{a}.\end{cases}$$

Начав с буквы a, мы получим список слов Фибоначчи $\varphi^k(\mathtt{a})$, $k \geq 0$, первые элементы которого приведены ниже:

$fib_0 = \varphi^0(\mathtt{a}) = \mathtt{a}$
$fib_1 = \varphi^1(\mathtt{a}) = \mathtt{ab}$
$fib_2 = \varphi^2(\mathtt{a}) = \mathtt{aba}$
$fib_3 = \varphi^3(\mathtt{a}) = \mathtt{abaab}$
$fib_4 = \varphi^4(\mathtt{a}) = \mathtt{abaababa}$
$fib_5 = \varphi^5(\mathtt{a}) = \mathtt{abaababaabaab}$
$fib_6 = \varphi^6(\mathtt{a}) = \mathtt{abaababaabaababaababa}$

В пределе получается такое ассоциированное бесконечное слово:

$$\mathbf{f} = \lim_{k\to\infty}\varphi^k(\mathtt{a}) = \mathtt{abaababaabaababaababaabaababaabaab}\cdots.$$

Эквивалентное определение слов Фибоначчи дает рекуррентное соотношение

$$\begin{cases}fib_0 = \mathtt{a},\\ fib_1 = \mathtt{ab},\\ fib_{k+1} = fib_k fib_{k-1}, \quad k \geq 1.\end{cases}$$

Длины этих слов образуют последовательность чисел Фибоначчи, т. е. $|fib_k| = F_{k+2}$. Напомним, что ***числа Фибоначчи*** определяются рекуррентно следующим образом:

$$\begin{cases}F_0 = 0,\\ F_1 = 1,\\ F_{k+1} = F_k + F_{k-1}, \quad k \geq 1.\end{cases}$$

Из их многочисленных свойств отметим следующие:

- $\gcd(F_n, F_{n-1}) = 1$ для $n \geq 2$,
- F_n – ближайшее целое к $\Phi^n/\sqrt{5}$, где $\Phi = ½(1 + \sqrt{5}) = 1.61803\ldots$ – ***золотое сечение***.

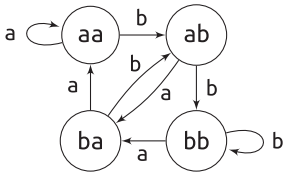
Интерес к словам Фибоначчи проистекает из их комбинаторных свойств и большого число содержащихся в них повторений. Однако бесконечное число Фибоначчи не содержит ни одного фактора с показателем степени большим, чем $\Phi^2 + 1 = 3.61803\ldots$.

Слова де Брёйна определяются над алфавитом $A = \{\mathtt{a},\mathtt{b}\}$ и параметризуются целым положительным числом k. Слово $x \in A^+$ называется словом де Брёйна порядка k, если каждое слово из A^k входит в x ровно один раз. Например, `ab` и `ba` – единственные слова де Брёйна порядка 1. Другой пример: `aaababbbaa` является словом де Брёйна порядка 3, поскольку восемь его факторов длины 3 совпадают с восемью словами A^3: `aaa`, `aab`, `aba`, `abb`, `baa`, `bab`, `bba` и `bbb`.

Существование слов де Брёйна порядка $k \geq 2$ можно проверить с помощью автомата де Брёйна, определенного следующим образом:

- состояниями являются слова из A^{k-1};
- ребра имеют вид (av, b, vb), где $a, b \in A$ и $v \in A^{k-2}$.

На рисунке ниже изображен автомат для слов де Брёйна порядка 3. Заметим, что из каждого состояния выходит ровно два ребра, помеченных буквами `a` и `b`, и что в каждое состояние входит ровно два ребра, помеченных одной и той же буквой. Поэтому ассоциированный с автоматом граф удовлетворяет условию Эйлера: все вершины имеют четную степень. Это означает, что существует эйлеров обход графа. Буквы, помечающие ребра, вошедшие в этот обход, образуют ***круговое слово де Брёйна***. Добавление в его конец его же префикса длины $k - 1$ дает обыкновенное слово де Брёйна.



Можно также доказать, что количество слов де Брёйна порядка k экспоненциально зависит от k.

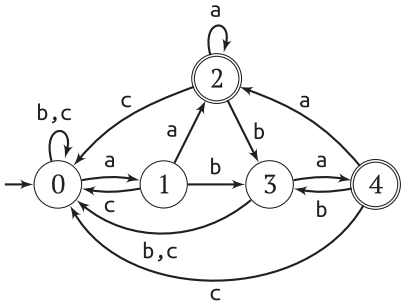
Слова де Брёйна можно определить и над большим алфавитом, они часто используются в качестве примеров предельных случаев, потому что содержат все факторы заданной длины.

Автоматы

Конечный ***автомат*** M над конечным алфавитом A состоит из следующих компонентов: конечное множество ***состояний*** Q, ***начальное*** состояние q_0,

множество ***заключительных*** состояний $T \subseteq Q$ и множество $F \subseteq Q \times A \times Q$ ***помеченных ребер***, или ***дуг***[1], соответствующих ***переходам*** состояний. Будем обозначать автомат M четверкой (Q, q_0, T, F) или иногда просто (Q, F), например если q_0 неявно подразумевается, а $T = Q$. Говорят, что дуга (p, a, q) исходит из состояния p и входит в состояние q, при этом состояние p называется ***исходным***, буква a – ***меткой*** дуги, а состояние q – ***конечным***. На рисунке ниже приведено графическое изображение автомата.

Количество дуг, исходящих из данного состояния, называется ***полустепенью исхода*** этого состояния. ***Полустепень захода*** состояния определяется аналогично. По аналогии с графами состояние q называется ***преемником*** состояния p по букве a, если $(p, a, q) \in F$; будем также говорить, что пара (a, q) является ***помеченным преемником*** состояния p.



Путем длины n в автомате $M = (Q, q_0, T, F)$ называется последовательность n соседних дуг $\langle (p_0, a_0, p'_0), (p_1, a_1, p'_1), \ldots, (p_{n-1}, a_{n-1}, p'_{n-1}) \rangle$ такая, что $p'_k = p_{k+1}$ для $k = 0, 1, \ldots, n-2$. ***Меткой*** пути называется слово $a_0 a \ldots a_{n-1}$, его ***началом*** – состояние p_0, а ***концом*** – состояние p'_{n-1}. Путь в автомате M называется ***успешным***, если его началом является начальное состояние q_0, а конец принадлежит T. Говорят, что автомат ***распознает***, или ***допускает***, слово, если оно является меткой успешного пути. Язык, состоящий из слов, распознаваемых автоматом M, обозначается $Lang(M)$.

Автомат $M = (Q, q_0, T, F)$ называется ***детерминированным***, если для любой пары $(p, a) \in Q \times A$ существует не более одного состояния $q \in Q$ для каждой дуги $(p, a, q) \in F$. В таком случае естественно рассмотреть ***функцию переходов*** автомата $\delta : Q \times A \to Q$, определенную для каждой дуги $(p, a, q) \in F$ так, что $\delta(p, a) = q$, и не определенную больше нигде. Функция δ тривиально распространяется на слова.

Известно, что любой язык, допускаемый автоматом, допускается также детерминированным автоматом и что существует единственный (с точностью до именования состояний) минимальный детерминированный автомат, допускающий язык.

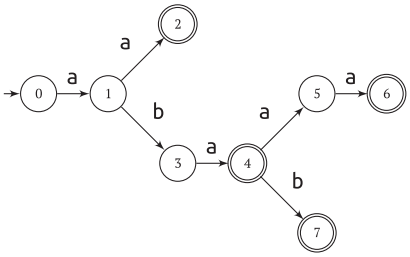
---

[1] Принято называть ребра ориентированного графа дугами, мы будем далее придерживаться этой терминологии. – *Прим. перев.*

Префиксные деревья

Префиксное дерево (trie) $\mathcal{T}$ над алфавитом A – это автомат, для которого пути, исходящие из начального состояния, корня, не сходятся. Чаще всего префиксные деревья используются для представления конечных множеств слов. Если никакое слово не является префиксом никакого другого слова из данного множества, то все слова будут ассоциированы с листьями дерева.

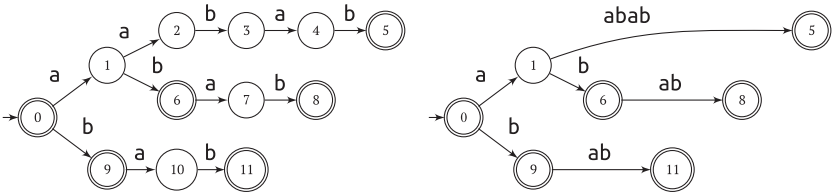
На рисунке ниже изображено префиксное дерево $\mathcal{T}$({aa, aba, abaaa, abab}). Состояния соответствуют префиксам слов, принадлежащих множеству. Например, состояние 3 соответствует префиксу длины 2 слов abaaa и abab. Заключительные состояния (обведенные двойной окружностью) 2, 4, 6 и 7 соответствуют словам из множества.



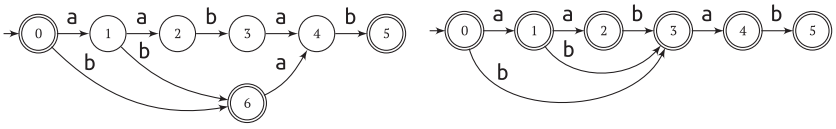
Суффиксные структуры

Суффиксные структуры данных, в которых хранятся суффиксы слова, важны для построения эффективных индексов. В этом качестве можно использовать префиксные деревья, но их размер может расти квадратично. Одно из решений этой проблемы – уплотнить префиксное дерево, получив суффиксное дерево слова. Для этого удаляются нелистовые узлы, из которых исходит только одно ребро, а дуги помечаются соответственными факторами слова. Удаленные узлы иногда называют неявными узлами суффиксного дерева, а оставшиеся узлы – явными.

Ниже показано префиксное дерево $\mathcal{T}$(*Suff*(aabab)) суффиксов aabab (слева) и его ***суффиксное дерево*** $\mathcal{ST}$(aabab) (справа). Чтобы получить полную структуру линейного размера, каждый фактор слова, помечающий дугу, нужно представить парой целых чисел (позиция, длина).



Второе решение, позволяющее уменьшить размер префиксного дерева суффиксов, – минимизировать его, т. е. рассмотреть минимальный детерминированный автомат, допускающий суффиксы слова, его ***суффиксный автомат***. На рисунке ниже слева показан суффиксный автомат $\mathcal{S}$(aabab) слова aabab.



Известно, что в $\mathcal{S}(x)$ менее $2|x|$ состояний и менее $3|x|$ дуг, т. е. его общий размер $O(|x|)$ линейно зависит от $|x|$. Факторный автомат $\mathcal{F}(x)$ слова, т. е. минимальный детерминированный автомат, допускающий его факторы, может быть еще меньше, потому что все его состояния заключительные. На рисунке выше справа показан факторный автомат слова aabab, в котором состояние 6 автомата $\mathcal{S}$(aabab) объединено с состоянием 3.

Суффиксный массив

Суффиксный массив слова также применяется для построения индексов, но работает иначе, чем деревья или автоматы. Его основное назначение – отсортировать суффиксы слова, чтобы сделать возможным двоичный поиск по его факторам. Чтобы повысить эффективность поиска, вводится в рассмотрение еще одно понятие: наибольший общий префикс соседних суффиксов в отсортированном списке.

Эта информация хранится в двух массивах: SA и LCP. Массив SA получен обращением массива Rank, который содержит ранги (т. е. места в отсортированном списке) суффиксов с привязкой к их начальным позициям.

Таблицы ниже построены для слова aababa. Его отсортированный список суффиксов имеет вид a, aababa, aba, ababa, ba, baba, а их начальные позиции равны соответственно 5, 0, 3, 1, 4 и 2. Этот последний список, индексированный рангами суффиксов, и хранится в массиве SA.

i	0	1	2	3	4	5
$x[i]$	a	a	b	a	b	a
Rank[i]	1	3	5	2	4	0

r	0	1	2	3	4	5	6	7	8	9	10	11	12
SA[r]	5	0	3	1	4	2							
LCP[r]	0	1	1	3	0	2	0	0	1	0	0	0	0

Таблица LCP содержит ***наибольшие общие префиксы***, которые хранятся как максимальные длины общих префиксов соседних суффиксов:

$$\text{LCP}[r] = |lcp(x[\text{SA}[r-1]..|x|-1], x[\text{SA}[r]..|x|-1])|,$$

где *lcp* обозначает наибольший общий префикс двух слов. Она, например, дает LCP[0..6]. Следующие значения в диапазоне LCP[7..12] соответствуют той же информации для суффиксов, начинающихся в позициях d и f, когда пара (d, f) встречается при двоичном поиске. Формально для такой пары значение хранится в позиции $|x| + 1 + \lfloor (d + f)/2 \rfloor$. Например, в приведенном выше массиве LCP значение 1, соответствующее паре (0, 2), максимальной длине префиксов между $x[5..5]$ и $x[3..5]$, хранится в позиции 8.

Таблица Rank в основном применяется в приложениях суффиксного массива, не связанных с поиском.

Сжатие

Самые эффективные методы ***сжатия*** текстов общего вида основаны либо на алгоритме факторизации слов Лемпеля–Зива, либо на более простых способах, опирающихся на преобразование слов Барроуза–Уилера.

При обработке слова в онлайновом режиме цель ***схемы сжатия Лемпеля–Зива*** заключается в том, чтобы уловить информацию, которая уже встречалась раньше. В результате имеем факторизацию слова x вида $u_0u_1...u_k$, где u_i – самый длинный префикс $u_i...u_k$, который встречался до появления x. Если этот префикс пуст, то выбирается первая буква $u_i...u_k$, которая не встречается в $u_0...u_{i-1}$. Допуская вольность речи, фактор u_i иногда называют ***наибольшим предыдущим фактором*** в позиции $|u_0...u_{i-1}|$ слова x.

Например, слово `abaababababaaabab` факторизуется следующим образом: `a·b·a·aba·baba·aabab·b`.

Существует несколько способов определить факторы, составляющие разложение, опишем некоторые из них. Фактор u_i может включать букву, которая следует сразу за вхождением наибольшего предыдущего фактора в позиции $|u_0...u_{i-1}|$; это сводится к расширению фактора, встречавшегося ранее. Предыдущие вхождения факторов можно выбирать из факторов u_0, ..., u_{i-1}, или из всех факторов слова $u_0...u_{i-1}$ (чтобы избежать перекрытия вхождений), или из всех факторов, встречавшихся прежде. В результате получается широкое разнообразие программ сжатия текста, основанных на этом методе.

При проектировании алгоритмов обработки слов факторизация используется также с целью уменьшить объем онлайновой обработки, сохранив то, что уже было сделано при рассмотрении предыдущих вхождений факторов.

Преобразование Барроуза–Уилера слова x – это обратимое отображение, которое переводит $x \in A^k$ в $\mathrm{BW}(x) \in A^k$. Смысл его в том, чтобы сгруппировать буквы, имеющие одинаковый контекст в x. Кодирование производится следующим образом. Рассмотрим отсортированный список ротаций x (сопряженных слов). Тогда $\mathrm{BW}(x)$ – это слово, составленное из последних букв отсортированных ротаций, находящихся в последнем столбце соответствующей таблицы.

Например, ротации слова `banana` показаны на рисунке ниже слева, а отсортированный список ротаций – справа. Таким образом, BW(`banana`) = `nnbaaa`.

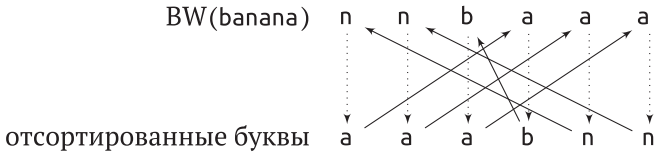
0	b	a	n	a	n	a	5	a	b	a	n	a	n
1	a	n	a	n	a	b	3	a	n	a	b	a	n
2	n	a	n	a	b	a	1	a	n	a	n	a	b
3	a	n	a	b	a	n	0	b	a	n	a	n	a
4	n	a	b	a	n	a	4	n	a	b	a	n	a
5	a	b	a	n	a	n	2	n	a	n	a	b	a

Это отображение переводит сопряженные слова в одно. Если выбирать слово Линдона в качестве представителя класса примитивного слова, то отображение становится биективным. Чтобы восстановить исходное слово x, отличное от слова Линдона, достаточно сохранить позицию первой буквы x в BW(x).

Главное свойство этого преобразования состоит в том, что относительный порядок вхождений данной буквы в BW(x) и в отсортированный список всех букв одинаков. Это позволяет декодировать BW(x).

Чтобы проделать это для слова `nnbaaa` из примера выше, сначала отсортируем буквы – получится `aaabnn`. Зная, что первая буква исходного слова находится в позиции 2 слова `nnbaaa`, мы можем начать декодирование: первой буквой является `b`, за ней следует буква `a` в той же позиции 2 слова `aaabnn`. Это третье вхождение `a` в слово `aaabnn`, соответствующее ее третьему вхождению в слово `nnbaaa`, за которым следует `n`. И так далее.

Процесс декодирования похож на обход цикла в графе ниже, начиная с правильной буквы. Если начать с другой буквы, то получится слово, сопряженное начальному.



Соглашения о псевдокоде алгоритмов

Применяемый нами стиль языка описания алгоритмов близок к реальным языкам программирования, но уровень абстракции выше. Мы придерживаемся следующих соглашений:

- отступами обозначается блочная структура составных предложений;
- строки кода нумеруются, чтобы на них можно было сослаться в тексте;
- символом ▷ начинается комментарий;
- чтобы обозначить доступ к атрибуту объекта, указывается имя атрибута, а за ним в квадратных скобках – идентификатор, ассоциированный с объектом;
- переменная, представляющая объект (таблицу, очередь, дерево, слово, автомат), является указателем на этот объект;
- аргументы передаются процедурам и функциям по значению;

- переменные, употребляемые внутри процедур и функций, считаются локальными, если явно не оговорено противное;
- булевы выражения вычисляются лениво слева направо;
- запись вида $(m_1, m_2, \ldots) \leftarrow (exp_1, exp_2, \ldots)$ – сокращенное обозначение последовательности присваиваний $m_1 \leftarrow exp_1$, $m_2 \leftarrow exp_2$, ...

В качестве примера ниже описан алгоритм Trie. Он порождает префиксное дерево по конечному множеству слов (словарю) X. В цикле **for**, занимающем строки 2–10, рассматривается каждое слово из X, а в цикле **for** в строках 4–9 буквы этого слова вставляются в структуру данных одна за другой. По завершении внутреннего цикла последнее рассмотренное состояние t, завершающее путь из начального состояния и помеченное текущим словом, делается заключительным (строка 10).

```
Trie(X конечное множество слов)
 1  M ← New-automaton()
 2  for каждой строки x ∈ X do
 3      t ← initial(M)
 4      for каждой буквы a слова x последовательно do
 5          p ← Target(t, a)
 6          if p = nil then
 7              p ← New-state()
 8              Succ[t] ←Succ[t] ∪ {(a, p)}
 9          t ← p
10      terminal[t] ← true
11  return M
```

Примечания

При изложении основных понятий, связанных со словами, мы следовали работе [74]. Этот материал можно найти и в других учебниках по алгоритмам обработки текста, например Crochemore and Rytter [96], Gusfield [134], Crochemore and Rytter [98] и Smyth [228], а также в книгах, посвященных более широкой теме комбинаторики слов, например Lothaire [175–177], или в пособии Berstel and Karhumäki [34].