# Построение суффиксного дерева за линейное время
# Лекция № 1 курса «Алгоритмы для Интернета»

Юрий Лифшиц*

28 сентября 2006 г.

## Содержание



## 1. Введение в суффиксные деревья

### 1.1. Определение суффиксного дерева

Будем называть *текстом* $T$ строку из $n$ символов $t_1 \dots t_n$, а каждое окончание текста $t_i \dots t_n$ — его *суффиксом*.

*Суффиксное дерево* (ST) — это способ представления текста. Неформально говоря, чтобы построить ST для текста $T = t_1 \dots t_n$, нужно приписать специальный символ $ в конец текста, взять все $n + 1$ суффиксов, подвесить их за начала и склеить все ветки, идущие по одинаковым буквам. В каждом листе записывается номер суффикса, заканчивающегося в этом листе. Номером суффикса является индекс его начала в тексте $T$.

Заметим, что ни один суффикс в ST не может полностью лежать в другом суффиксе, поскольку они заканчиваются специальным символом $. Таким образом, листьев в ST всегда будет $n + 1$ для строки $t_1 \dots t_n$, то есть столько же, сколько суффиксов. Но общее число вершин в суффиксном дереве квадратично.

Разберемся теперь, как хранить суффиксное дерево, используя линейную память. Для этого оставим в ST только вершины *разветвления*, то есть имеющие не менее двух детей. Вместо строки для ребра будем хранить ссылку на сегмент текста $T[i..j]$. В таком виде суффиксное дерево называется *сжатым*.

*Законспектировал Иван Лагунов.

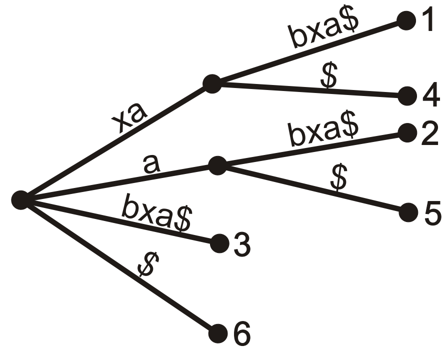


**Рис. 1.** Пример сжатого суффиксного дерева для строки xabxa$

Заметим, что, так как теперь каждая из внутренних вершин является вершиной разветвления, то она добавляет к своему поддереву как минимум один лист. Листьев же в ST всего $n+1$ для строки $t_1 \dots t_n$, поэтому внутренних вершин может быть в диапазоне $1 \dots n$.

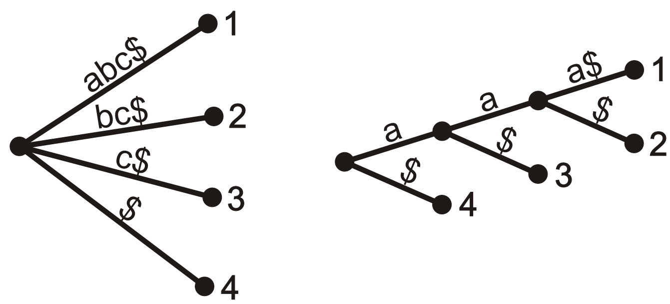


**Рис. 2.** Крайние случаи для числа внутренних вершин в сжатом суффиксном дереве: одна внутренняя вершина для строки abc$, три — для строки aaa$

Таким образом, всего вершин и ребер в сжатом суффиксном дереве будет линейное число, значит оно будет занимать линейную память.

## 1.2. Два применения

**Поиск подстрок**

Дан текст $T = t_1 \dots t_n$. Нужно так его «подготовить» за время $O(n)$, чтобы поиск любого шаблона $P$ занимал время $O(|P|)$. *Шаблоном* называем строку, которую хотим найти в тексте $T$.

Приведем решение с помощью суффиксного дерева. Сначала построим суффиксное дерево для текста $T$. Будем читать шаблон вдоль дерева от корня. Если в какой-то момент не сможем прочитать следующую букву шаблона, значит шаблон ни разу не встречался в тексте $T$. Допустим, что он встречался, тогда, прочитав его, приходим в вершину $v$ или останавливаемся на ребре. В случае остановки на ребре пройдем дальше от корня ST до первой вершины $v$. Далее прочитаем числа, записанные в листьях-потомках вершины $v$. Эти числа — номера суффиксов, начинающихся с шаблона $P$, а значит индексы вхождений $P$ в текст $T$.

Сложность этого алгоритма $O(|P| + |Output|)$, где $|Output|$ — число листов в поддереве вершины $v$.

Заметим, что первое вхождение шаблона можно найти за время $O(|P|)$, для этого нужно предварительно для каждой вершины разветвления запомнить номер первого листа в ее поддереве. Это можно

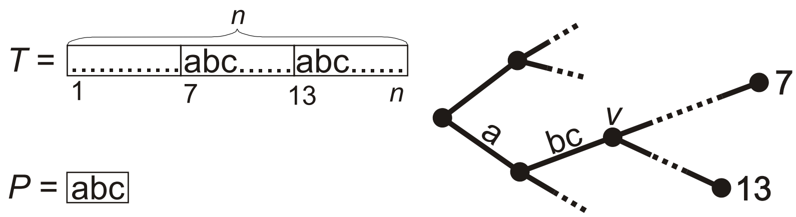


**Рис. 3.** Пример для задачи поиска шаблона $P$ в тексте $T$

сделать на этапе подготовки, например, обходом в глубину. Это очень важно, поскольку часто возникают задачи с очень большим текстом и короткими шаблонами для поиска.

В качестве примера можно привести трехтомный роман Л. Н. Толстого «Война и мир». Очевидно в данном случае, что невыгодно для каждого шаблона искать его вхождения в текст за время порядка длины текста. Поэтому построим суффиксное дерево для всего романа, найдем в нем для каждой внутренней вершины номер первого листа. Все это делается за время, линейное от длины романа. Далее сможем быстро обрабатывать запросы на поиск шаблонов в тексте.

Примерно так же действует программа Google Desktop, позволяющая искать шаблоны по всем текстам, хранящимся на компьютере. Она собирает все тексты, делает из них единую базу данных, похожую на суффиксное дерево, и сохраняет ее в специальном файле. При запросе на поиск программа читает шаблон в своем суффиксном дереве и выдает все файлы, содержащие этот шаблон. Конечно, при этом используются различные техники обрубания для суффиксного дерева, иначе его размер был бы сопоставим с размером всего файлового пространства на диске.

**Наибольшая общая подстрока**

Даны тексты $T_1$ и $T_2$. Требуется найти длину их наибольшей общей подстроки.

Для начала рассмотрим самый простой алгоритм, решающий эту задачу. В этом случае перебираем длину наибольшей общей подстроки, ее начала в текстах $T_1$ и $T_2$ и просто сравниваем подстроки. Тогда время работы алгоритма будет $O(n^4)$, где $n$ — максимум из длин текстов.

Опишем решение с помощью суффиксного дерева. Построим суффиксное дерево для конкатенации исходных текстов $T = T_1T_2$. Также для удобства можно между этими текстами вставить еще один специальный символ, но можно обойтись и без него. Назовем «*длинными*» и «*короткими*» суффиксами текста $T$ такие суффиксы, которые начинаются в текстах $T_1$ и $T_2$ соответственно. Для каждой внутренней вершины выясним: есть ли у нее одновременно потомки, соответствующие «короткому» и «длинному» суффиксам. Это можно сделать обходом в глубину. Для листа можно выяснить тип заканчивающегося в нем суффикса, зная индекс начала суффикса в тексте $T$. Поэтому можно узнать типы суффиксов для всех вершин обходом в глубину, при возвращении сохраняя для каждой родительской вершины типы суффиксов, которые хранились для ее детей.

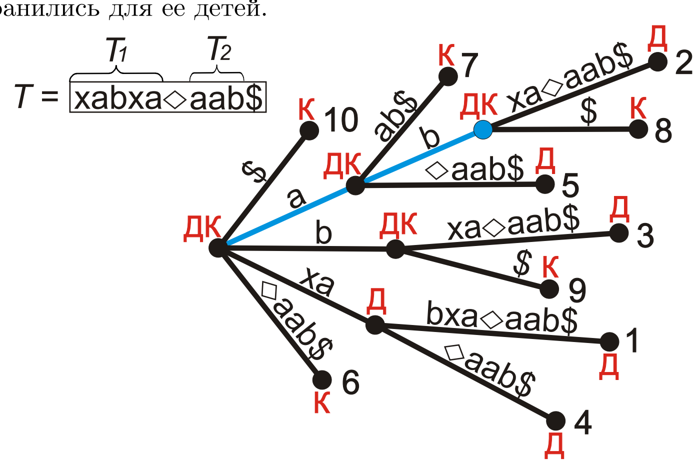


**Рис. 4.** Наибольшая общая подстрока текстов xabxa и aab равна ab

На рисунке 4 изображен пример, на котором буквы Д и К у вершины обозначают, что в ее поддереве заканчиваются «длинный» и «короткий» суффиксы соответственно. Если для вершины хранятся оба типа суффиксов (ДК), значит строка, соответствующая этой вершине в ST, встречается в $T$ как минимум в двух местах, начинаясь в $T_1$ и $T_2$. Найдем самую удаленную от корня такую вершину. Ее глубина — ответ для задачи. Сложность этого алгоритма $O(|T_1| + |T_2|)$.

## 1.3. Наивный кубический алгоритм

**On-line подход**

Этот подход основан на том, что данные на вход алгоритму подаются частями, будь то текст или какие-то запросы. Алгоритм, использующий on-line подход, читает последовательно поступающие данные и получает готовое решение на каждом шаге.

Рассмотрим теперь on-line подход для задачи построения суффиксного дерева. Для каждого суффикса его дополнение до исходного текста $T = t_1 \dots t_n$ будем называть *префиксом*, то есть *префикс* — любое начало текста $t_1 \dots t_i$. Будем строить ST не только для всего текста, но последовательно для всех его префиксов:

0. Строим суффиксное дерево для $t_1$.

1. Расширяем его до дерева для $t_1 t_2$.

$\dots$

$n-1$. Расширяем дерево для $t_1 \dots t_{n-1}$ до дерева для $t_1 \dots t_n$.

$n$. Расширяем дерево для $t_1 \dots t_n$ до дерева для $t_1 \dots t_n\$$.

Каждый шаг этого списка будем называть *фазой* алгоритма.

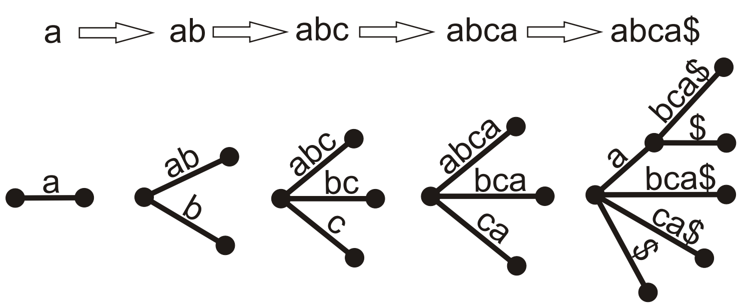


**Рис. 5.** Последовательность фаз алгоритма для строки abca$. На промежуточных фазах используются неявные суффиксные деревья, которые поясняются далее

Вернемся к примеру с «Войной и миром». Представим, что нам не будет дан сразу весь роман, а будут даваться по очереди первый, второй и третий тома. Тогда можно построить суффиксное дерево сначала по первому тому, потом достроить его для двух томов и, наконец, для всего романа.

**Неявные суффиксные деревья**

*Неявное суффиксное дерево* (IST) — это суффиксное дерево для текста без \$ на конце. Некоторые суффиксы текста в нем заканчиваются не в листьях, и их номер нигде не хранится.

На рисунке 6 видно, что в явном суффиксном дереве добавилось несколько веток. Это объясняется тем, что к строке xabxa добавляется не встречавшийся ранее специальный символ \$. В результате, для каждого суффикса происходит либо ответвление с возникновением новой ветки, либо продление символом \$.

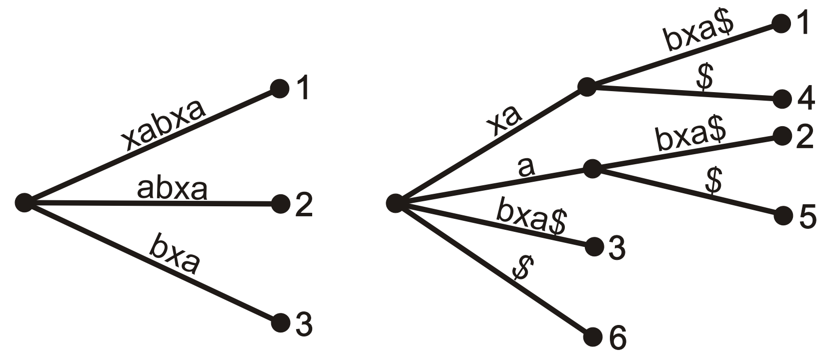


**Рис. 6.** Неявное суффиксное дерево для строки xabxa и явное суффиксное дерево для строки xabxa\$

**Фаза = последовательность продлений**

Рассмотрим фазу $i$. В ней мы перестраиваем IST для $t_1 \dots t_i$ в IST для $t_1 \dots t_i t_{i+1}$. Для каждого $j$ от 1 до $i$ находим в суффиксном дереве конец суффикса $t_j \dots t_i$. Далее *продляем* его буквой $t_{i+1}$, если необходимо.

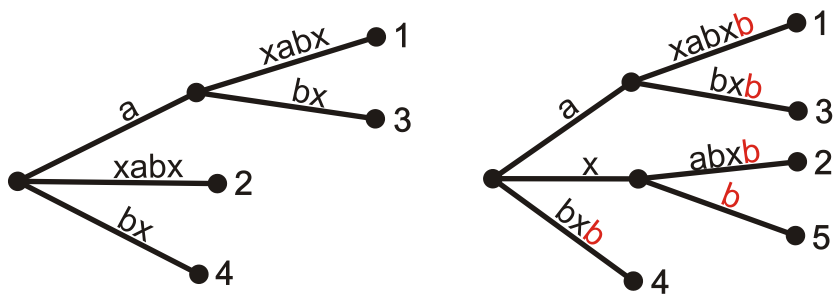


**Рис. 7.** Пример фазы: из суффиксного дерева для строки axabx получаем дерево для строки axabxb

Например, рассмотрим фазу на рисунке 7. Надо найти концы суффиксов строки axabx, для этого просто по очереди будем читать их от корня суффиксного дерева. Читаем первый суффикс axabx, приходим в лист 1, продляем его, дальше читаем суффикс xabx, приходим в лист 2, продляем, и так далее для всех суффиксов. Но ситуации могут отличаться, как, например, для суффикса x, когда, прочитав его, остаемся на ребре. В этом случае создаем новую ветку и лист 5.

Всего может быть три типа продлений. Рассмотрим возможные варианты.

1. Продление листа.

   Эта ситуация возникает, когда, прочитав суффикс $t_j \dots t_i$, мы пришли в лист суффиксного дерева. Тогда «удлиняем» ребро, ведущее в лист, добавляя к строке, соответствующей ребру, новую букву $t_{i+1}$.

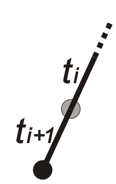


**Рис. 8.** Продление листа буквой $t_{i+1}$

2. Ответвление буквы.

   Прочитав суффикс $t_j \dots t_i$, мы могли остановиться не в листе, а во внутренней вершине или даже на каком-то ребре.

   (a) В случае, когда остановились во внутренней вершине $v$ и из нее нет исходящего ребра по букве $t_{i+1}$, добавляем новое ребро из $v$ в новый лист и записывает на ребре букву $t_{i+1}$.

   (b) Но мы могли остановиться на ребре, потому что ребру соответствует сегмент текста $T[k..m] = t_k \dots t_m$, а не одна буква. Этот случай изображен на рисунке 9.

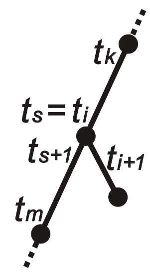


**Рис. 9.** Ответвление буквы $t_{i+1}$

   Допустим, мы прочитали на ребре только часть текста $t_k \dots t_s$, которая совпадает с подсуффиксом нашего суффикса $t_j \dots t_i$, где $t_s = t_i$. Тогда если $t_{s+1} \neq t_{i+1}$, разобьем новой внутренней вершиной $v$ это ребро на два, соответствующие строкам $T[k..s]$ и $T[s+1..m]$. Затем создадим новое ребро с буквой $t_{i+1}$ из вершины $v$ в новый лист, как в случае 2(a).

3. Пустое правило.

   Если, прочитав суффикс $t_j \dots t_i$, мы остановились во внутренней вершине или на ребре (как в случае 2), но дальше уже есть буква $t_{i+1}$, не создаем ничего нового.

#### Кубическая оценка времени работы

Оценим время работы нашего алгоритма.

Всего $n$ фаз. В фазе $i$ продлеваем $i$ суффиксов. Причем, $j$-й суффикс в этой фазе $t_j \dots t_i$ имеет длину $i - j + 1$. Тогда продление $j$ в фазе $i$ требует время $O(i - j)$. Поэтому сначала суммируем по $j$, чтобы узнать стоимость фазы $i$. Далее суммируем по $i$, получая оценку времени работы всего алгоритма. Итого: $O(\sum_{i=1}^{n} \sum_{j=1}^{i} i - j) = O(n^3)$.

## 2. Квадратичный алгоритм

#### Идея вспомогательных данных

Вначале объясним стандартную идею из теории алгоритмов. Допустим, нужно вычислить массив $X_1, \dots, X_n$ по индукции. В таком случае для вычисления очередного элемента $X_{i+1}$ может не хватить знания элемента $X_i$, то есть могут потребоваться дополнительные данные. Поэтому иногда полезно определить вспомогательный массив $Y_1, \dots, Y_n$. Далее последовательно вычислять оба массива, то есть сначала вычислить $X_1$ и $Y_1$, с их помощью вычислить $X_2$, затем вычислить $Y_2$, и так далее. И, наконец, с помощью $X_{n-1}$ и $Y_{n-1}$ вычислить $X_n$.

Далее нам также потребуются вспомогательные данные, в качестве которых выступят суффиксные стрелки.

**Определение суффиксных стрелок**

Для каждой *внутренней* вершины, соответствующей суффиксу $t_1 \dots t_k$, нарисуем *суффиксную стрелку* в вершину, соответствующую суффиксу $t_2 \dots t_k$. Заметим, что этот суффикс будет заканчиваться именно в вершине, а не на ребре. Дело в том, что суффикс $t_1 \dots t_k$ заканчивается во внутренней вершине, значит он как подстрока уже встречался в исходном тексте как минимум с двумя различными продолжениями. Поэтому, если отбросить первую букву $t_1$, то строка $t_2 \dots t_k$ тоже встречалась как минимум с двумя различными продолжениями. Таким образом, суффикс $t_2 \dots t_k$ тоже будет заканчиваться во внутренней вершине.

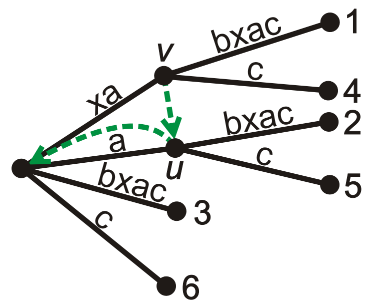


**Рис. 10.** Неявное суффиксное дерево для строки xabxac с суффиксными стрелками

Рассмотрим пример на рисунке 10. Возьмем вершину $v$, в которую приходим по строке xa. Чтобы провести суффиксную стрелку, отбросим первую букву x, осталась буква a. Прочитав ее от корня дерева, приходим в вершину $u$, это и есть адрес суффиксной стрелки. Теперь есть строка a, отбросим первую букву и получим пустую строку, которая соответствует корню дерева. Поэтому проводим вторую суффиксную стрелку в корень ST.

**Обновление суффиксных стрелок с опозданием**

Для начала отметим, что старые суффиксные стрелки сохраняются при продлениях суффиксов. Поэтому осталось понять, когда нужно рисовать *новую* суффиксную стрелку.

Рассмотрим случай на рисунке 11, когда мы применили продление по второму правилу (ответвление) к суффиксу $t_j \dots t_i$ и в дереве образовалась новая внутренняя вершина $v$. Тогда вторым ребенком вершины $v$ стал новый лист, на ребре к которому написана буква $t_{i+1}$. Тогда из $v$ нужно провести суффиксную стрелку.

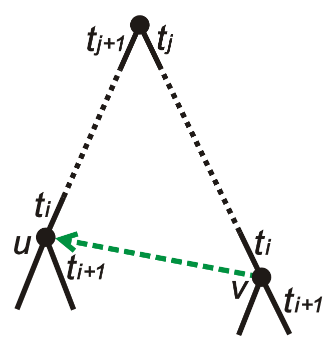


**Рис. 11.** Схема проведения суффиксных стрелок

Не будем отдельно искать место, куда должна указывать суффиксная стрелка. Просто перейдем к продлению следующего суффикса. Это будет суффикс $t_{j+1} \dots t_i$. Его конец и есть адрес суффиксной стрелки. Возможно, что этот суффикс так же, как и предыдущий, заканчивается на ребре. Но тогда снова будет продление по второму правилу, и образуется нужная нам внутренняя вершина $u$, в которую и проведем суффиксную стрелку из $v$.

**Фаза с прыжками**

Будем называть «*хвостом*» суффикса то место в суффиксном дереве, где мы остановимся, если будем читать суффикс от корня дерева. Таким образом, «хвост» может быть как в вершине дерева, так и на ребре.

Тогда можно сэкономить на нахождении всех «хвостов» внутри одной фазы. Пусть мы закончили работу с «хвостом» $j$-го суффикса $t_j \dots t_i$, то есть выполнили один из трех типов продлений. Найдем теперь хвост $(j+1)$-го суффикса с помощью техники «Вверх-Прыжок-Вниз».

Перейдем от $j$-го «хвоста» суффикса $t_j \dots t_i$ вверх до ближайшей внутренней вершины, соответствующей строке $t_j \dots t_k$. Из нее будет существовать суффиксная стрелка, так как для каждой новой внутренней вершины создается суффиксная стрелка в той же фазе. Прыгнем по этой стрелке в вершину, соответствующую строке $t_{j+1} \dots t_k$. Затем спустимся вниз по дереву, читая текст $t_{k+1} \dots t_i$. Таким образом, приходим к хвосту суффикса $t_{j+1} \dots t_i$.

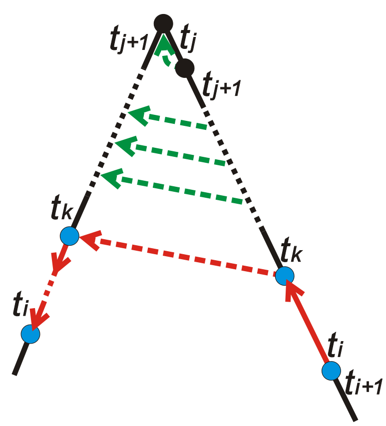


**Рис. 12.** Схема «Вверх-Прыжок-Вниз»

**Прыжки: подсчет высоты**

Будем называть *глубиной* вершины в дереве число ребер, по которым надо пройти из корня, чтобы попасть в вершину. Будем следить за глубиной указателя в дереве.

Сначала проходим по одному ребру вверх, уменьшая глубину вершины на 1. Далее переходим по суффиксной стрелке, глубина уменьшается не более чем на 1. Это объясняется тем, что на пути $A$ в дереве по суффиксу $t_{j+1} \dots t_k$ не более чем на одну вершину меньше, чем на пути $B$ по суффиксу $t_j \dots t_k$, потому что в пути $B$ каждой вершине $v$ будет соответствовать одна вершина $u$ пути $A$, в которую будет направлена суффиксная стрелка из $v$. Причем разница в одну вершину в этих путях может возникнуть, если в пути $B$ первому ребру будет соответствовать единственная буква $t_j$, тогда суффиксная стрелка из второй вершины после корня будет вести в корень.

Итак, после перехода вверх и прыжка по суффиксной стрелке глубина уменьшится не более чем на 2. Поэтому за фазу $i$ *вверх* мы совершили не более $2i$ переходов. Но заметим, что внутри одной фазы начальная глубина больше конечной, так как длины суффиксов в течение фазы уменьшаются до 1. Поэтому вниз мы не могли ходить больше, чем вверх. Таким образом, *вниз* мы тоже совершили не более $2i$ переходов. Поэтому общая оценка навигации внутри фазы составляет $4i$ переходов.

Поскольку всего $n$ фаз, в каждой из которых $O(n)$ переходов, общее время работы алгоритма стало $O(n^2)$.

## 3. Линейный алгоритм

**Анализ операций продления**

Итак, есть три типа продлений:

1. Удлинение: продление листа.
2. Ответвление буквы.
3. Пустое правило.

*Наблюдение 1:* как только мы применили пустое правило, дальше в фазе все продления — пустые.

Допустим, у нас есть «хвост» суффикса $\alpha = t_j \dots t_i$, и дальше в дереве уже есть переход по следующей букве $z = t_{i+1}$, но он мог появиться только при продлении идентичного текущему суффикса такой же буквой $z$.

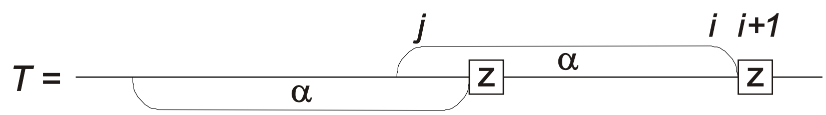


**Рис. 13.** Представление текста при продлении

В этом случае текст $T = t_1 \dots t_n$ представляется, как на рисунке 13. Значит, все последующие продления тоже уже выполнялись, поскольку любой суффикс строки $\alpha$ продлялся ранее буквой $z$. Поэтому все оставшиеся продления в этой фазе будут пустыми.

**«Живые» ребра**

*Наблюдение 2:* после того, как мы применили правило ответвления и создали новый лист, в следующих фазах к этому листу всегда будет применяться правило удлинения.

В связи с этим можно улучшить наш алгоритм. При создании нового листа будем кодировать новое ребро, ведущее в лист, как $T[i+1, x]$, где $x$ — указатель на переменную, хранящую конец текущего текста. Тогда все продления уже созданных листов можно произвести одной операцией $x := x + 1$. Заметим, что на этом ребре в дальнейшем могут происходить ответвления, тогда будет меняться начальный индекс $i + 1$, но не конечный.

**Модификация алгоритма**

Пусть «непустая часть» фазы $i-1$ закончилась на суффиксе $j^*$. Следовательно, к суффиксам $1, \dots, j^*$ применялись только правила 1 и 2, и каждый из них заканчивается в своем собственном листе. Опишем алгоритм для фазы $i$.

1. Присваиваем $x := x + 1$, одновременно продляя все суффиксы $1..j^*$.
2. Последовательно продляем суффиксы $j^* + 1, \dots, j'$, где $j'$ — первое применение пустого правила.
3. Обновляем номер суффикса, к которому последним применялось непустое правило $j^* := j' - 1$ и переходим к следующей фазе.

**Линейная оценка**

Работу алгоритма можно представить в виде схемы, показанной на рисунке 14.

По горизонтали у нас расположены суффиксы по увеличению номеров, а значит по порядку рассмотрения в алгоритме. Жирные зеленые линии обозначают участки индивидуальных продлений в каждой фазе, то есть это те суффиксы, которые продляются с ответвлением. Для всех суффиксов слева от зеленой

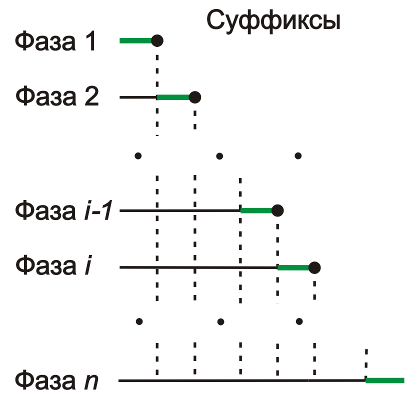


**Рис. 14.** Схема продлений суффиксов по фазам

линии происходит продление листа путем увеличения переменной $x := x+1$. Черные точки соответствуют моментам первого применения пустого правила в каждой фазе. Заметим, что на последней фазе не будет пустых продлений, потому что продляем невстречавшимся ранее специальным символом \$.

Оценим время работы алгоритма. Участки индивидуальных продлений по фазам перекрываются не более чем по одному суффиксу (суффикс, на котором первый раз применяется пустое правило, в следующей фазе снова обрабатывается индивидуально). Суммарное количество прыжков при продлениях линейно, как мы выяснили при анализе квадратичного алгоритма. Последняя фаза строит уже явное суффиксное дерево текста. Вот мы и получили оценку $O(n)$.

# Итоги

Суффиксное дерево — способ *представления* текста.

Применения: поиск подстрок, поиск наибольшей общей подстроки.

Основные идеи алгоритма: on-line построение, использование вспомогательных суффиксных стрелок, неравномерная оценка времени работы.

# Источники

[1] Pekka Kilpelainen. Lecture Slides
http://www.cs.uku.fi/~kilpelai/BSA05/lectures/print07.pdf

[2] Esko Ukkonen. On-line construction of suffix trees
http://www.cs.helsinki.fi/u/ukkonen/SuffixT1withFigs.pdf

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