Оптимизация алгоритма Риттера

Взгляд на сжатие как на предварительную обработку текста

Алексей Хворост, Иван Бурмистров

5 апреля 2012

С ростом размера входных данных для классических задач меняются алгоритмы, способные их эффективно решать.

Существует неколько подходов, например:

- Алгоритмы эффективного ввода вывода – алгоритмы, минимизирующие чтение данных с жесткого диска.
- Увеличение производительности за счет предварительной обработки.

С ростом размера входных данных для классических задач меняются алгоритмы, способные их эффективно решать.

Существует неколько подходов, например:

- Алгоритмы эффективного ввода вывода – алгоритмы, минимизирующие чтение данных с жесткого диска.
- Увеличение производительности за счет предварительной обработки.

С ростом размера входных данных для классических задач меняются алгоритмы, способные их эффективно решать.

Существует неколько подходов, например:

- Алгоритмы эффективного ввода вывода – алгоритмы, минимизирующие чтение данных с жесткого диска.
- Увеличение производительности за счет предварительной обработки.

Замечание

Предварительная обработка – это построение структур данных на основе исходной информации.

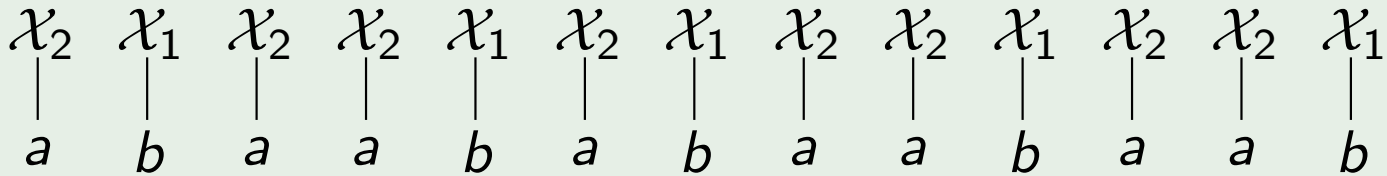
Примеры:

- Обратный индекс
- Суффиксные структуры (суффиксный массив, суффиксное дерево, суффиксный автомат)
- ...

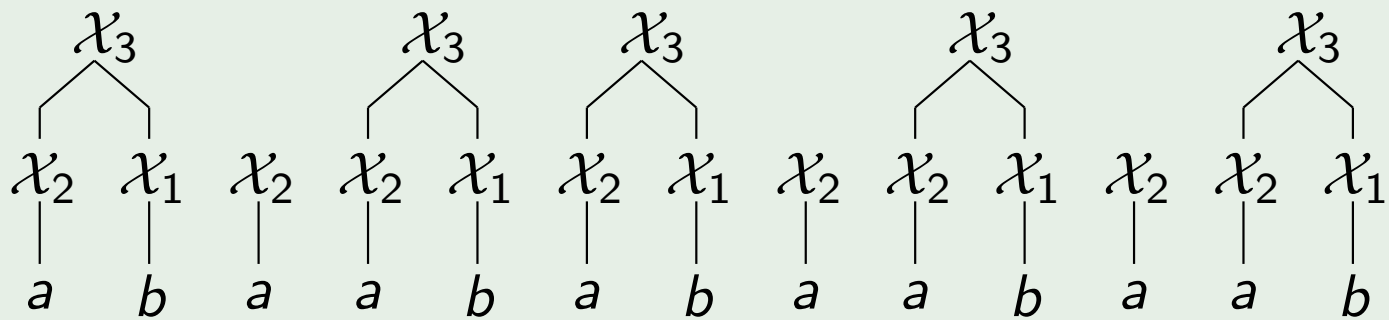
«Придумаем» структуру данных

$a \quad b \quad a \quad a \quad b \quad a \quad b \quad a \quad a \quad b \quad a \quad a \quad b$

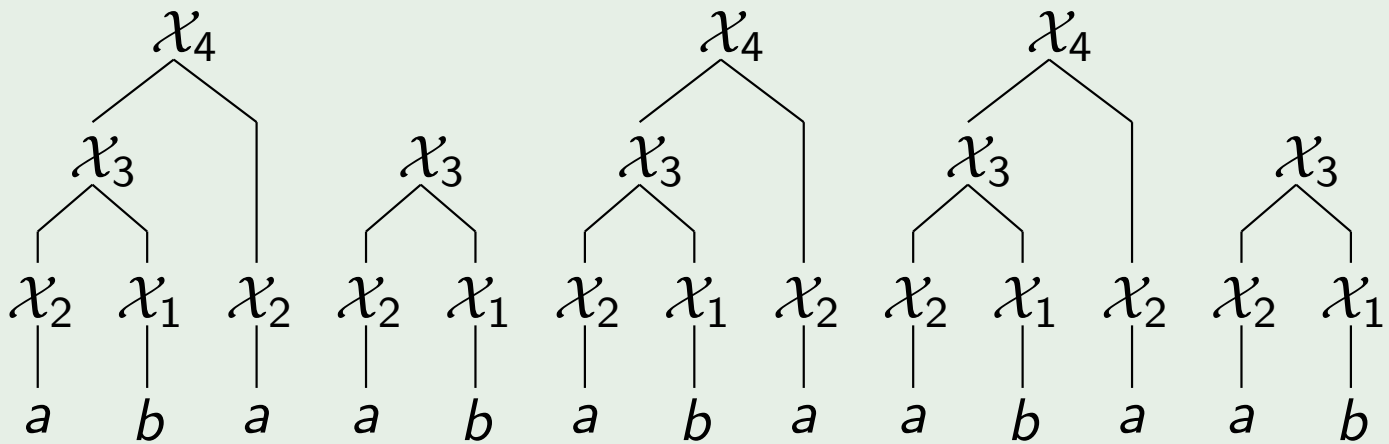
«Придумаем» структуру данных



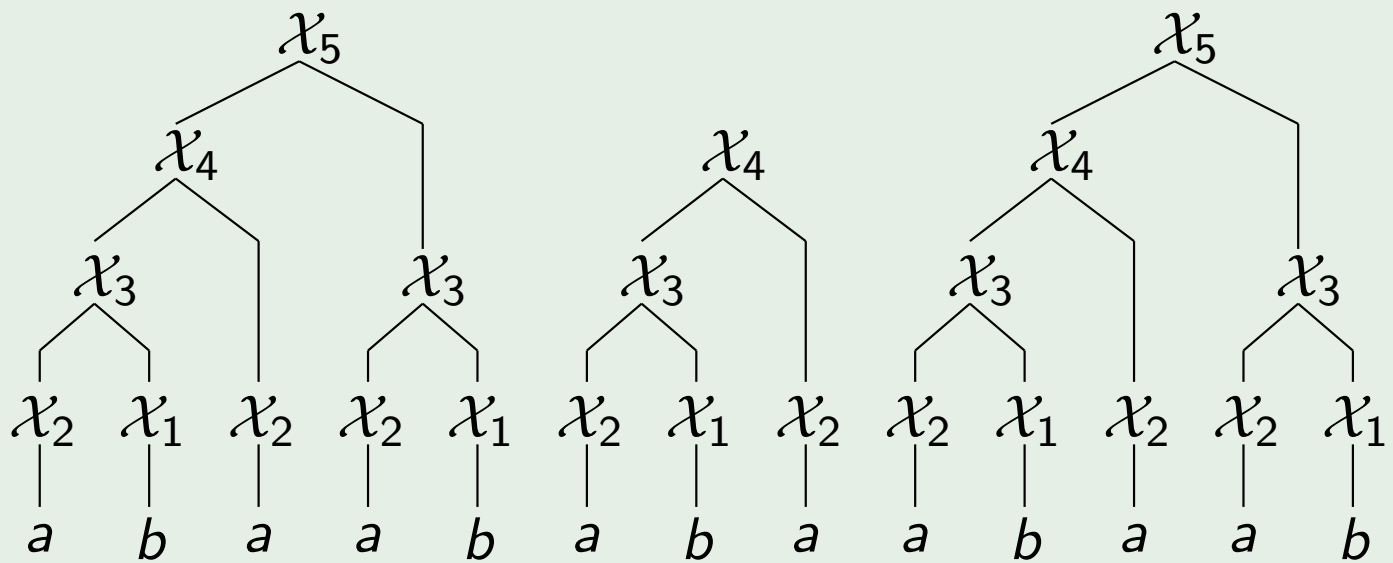
«Придумаем» структуру данных



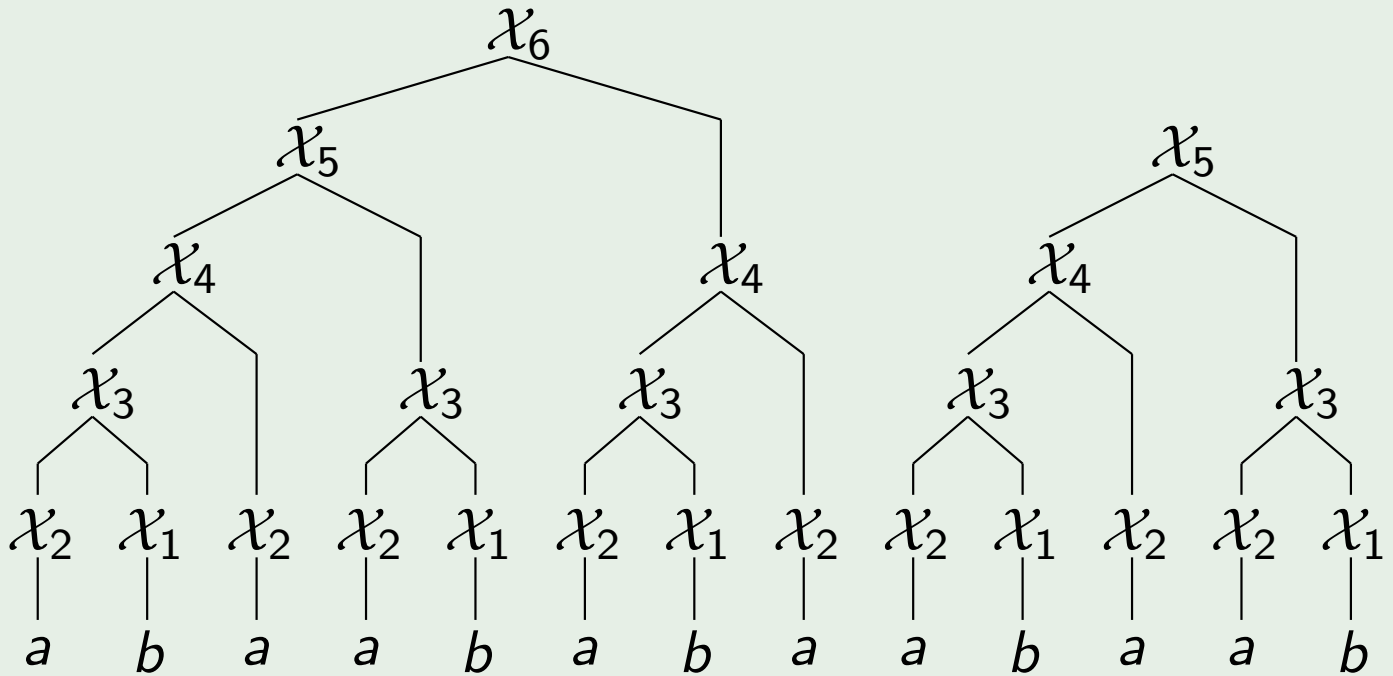
«Придумаем» структуру данных



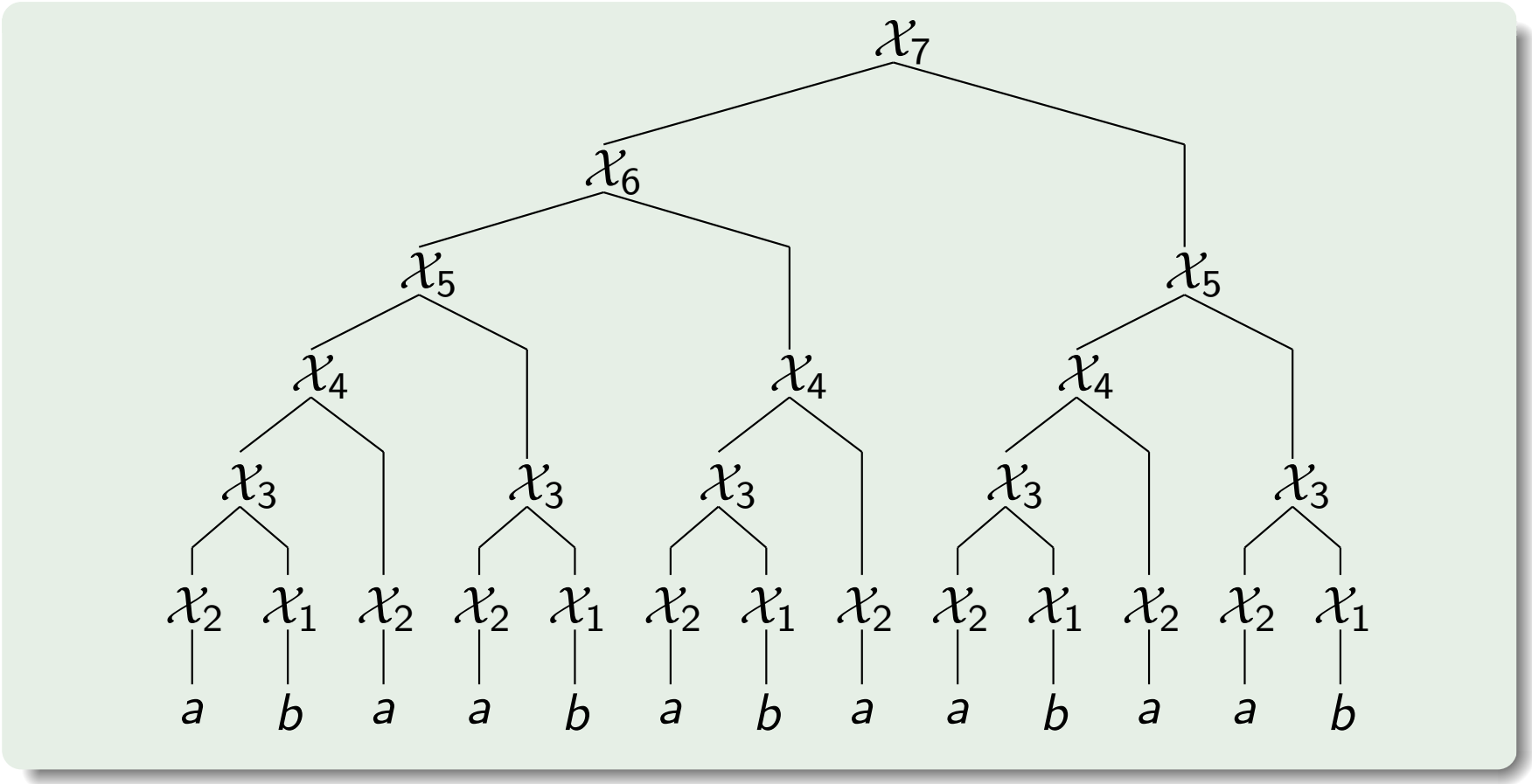
«Придумаем» структуру данных



«Придумаем» структуру данных



«Придумаем» структуру данных



Другое представление «придуманной» структуры данных

$$
\begin{aligned}
\mathcal{X}_1 &= b \\
\mathcal{X}_2 &= a \\
\mathcal{X}_3 &= \mathcal{X}_2 \cdot \mathcal{X}_1 \\
\mathcal{X}_4 &= \mathcal{X}_3 \cdot \mathcal{X}_2 \\
\mathcal{X}_5 &= \mathcal{X}_4 \cdot \mathcal{X}_3 \\
\mathcal{X}_6 &= \mathcal{X}_5 \cdot \mathcal{X}_4 \\
\mathcal{X}_7 &= \mathcal{X}_6 \cdot \mathcal{X}_5
\end{aligned}
$$

Для данного текста описанная структура является способом сжатия

Вывод

Сжатие – один из способов предварительной обработки, сохраняющий информацию о некоторых свойствах исходного текста

Для данного текста описанная структура является способом сжатия

Вывод

Сжатие – один из способов предварительной обработки, сохраняющий информацию о некоторых свойствах исходного текста

Определение

Прямолинейная программа (ПП) $\mathcal{X}$ размера n – это последовательность правил вывода

$$\mathcal{X}_1 = expr_1, \mathcal{X}_2 = expr_2, \ldots, \mathcal{X}_n = expr_n,$$

где $\mathcal{X}_i$ – это **переменные**, а $expr_i$ – это **выражения** вида:

- $expr_i$ – символ из алфавита Σ (**терминальные** правила).
- $expr_i = \mathcal{X}_l \cdot \mathcal{X}_r$ $(l, r < i)$ (**нетерминальные** правила).

Прямолинейная программа – это грамматика, выводящая в точности одно слово.

Определение

Прямолинейная программа (ПП) $\mathcal{X}$ размера n – это последовательность правил вывода

$$\mathcal{X}_1 = expr_1, \mathcal{X}_2 = expr_2, \ldots, \mathcal{X}_n = expr_n,$$

где $\mathcal{X}_i$ – это **переменные**, а $expr_i$ – это **выражения** вида:

- $expr_i$ – символ из алфавита Σ (**терминальные** правила).
- $expr_i = \mathcal{X}_l \cdot \mathcal{X}_r$ $(l, r < i)$ (**нетерминальные** правила).

Прямолинейная программа – это грамматика, выводящая в точности одно слово.

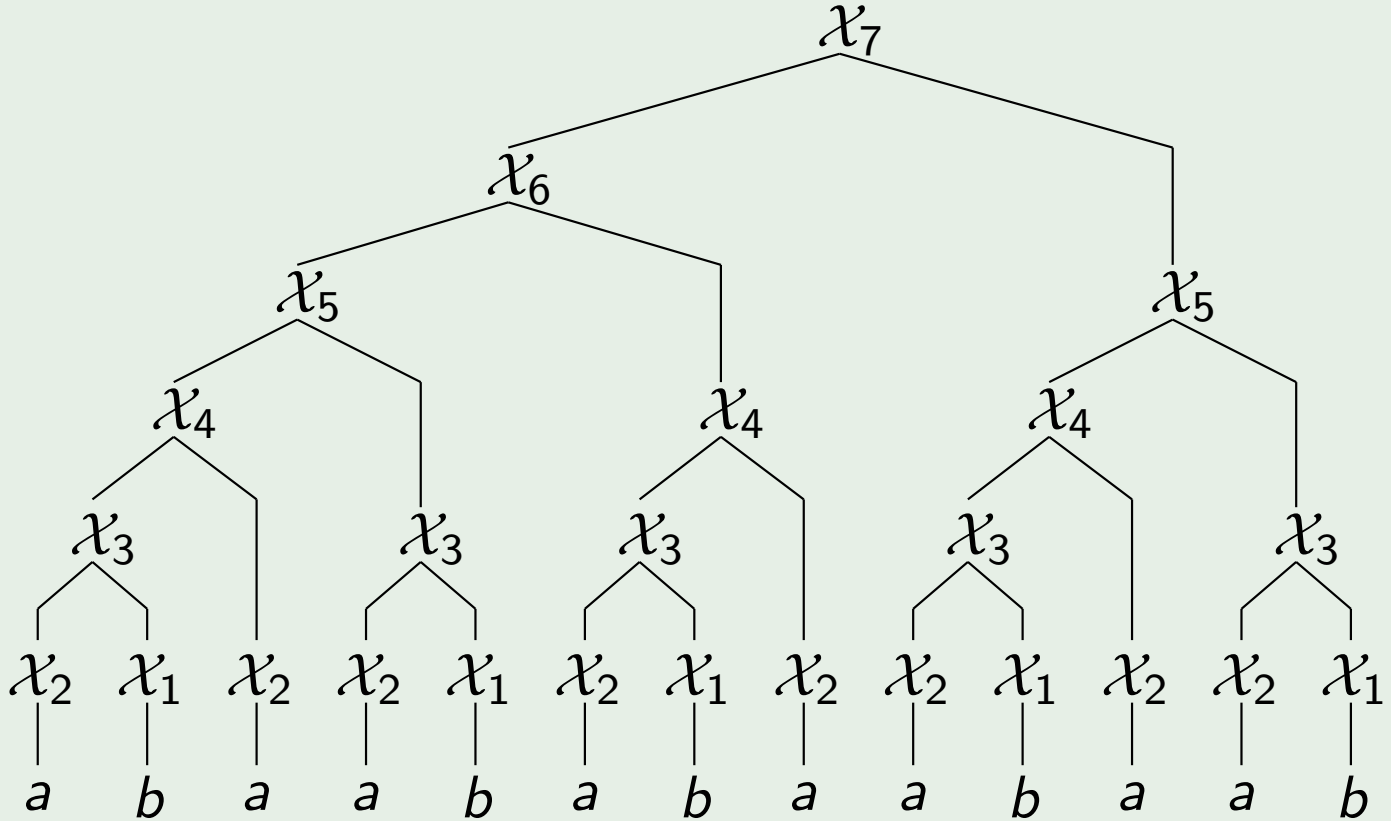
ПП $\mathcal{X}$, выводящая строку «*abaababaabaab*»

Пример

$$\begin{aligned}
\mathcal{X}_1 &= b \\
\mathcal{X}_2 &= a \\
\mathcal{X}_3 &= \mathcal{X}_2 \cdot \mathcal{X}_1 \\
\mathcal{X}_4 &= \mathcal{X}_3 \cdot \mathcal{X}_2 \\
\mathcal{X}_5 &= \mathcal{X}_4 \cdot \mathcal{X}_3 \\
\mathcal{X}_6 &= \mathcal{X}_5 \cdot \mathcal{X}_4 \\
\mathcal{X}_7 &= \mathcal{X}_6 \cdot \mathcal{X}_5
\end{aligned}$$

Дерево вывода $\mathcal{X}$

Пример



Теорема

Пусть $\mathcal{T}$ – ПП размера n, выводящая строку T, $\mathcal{P}$ – ПП размера m, выводящая строку P.
Алгоритм Лившица позволяет найти все вхождения текста P в текст T за время $O(n^2 m)$.

Пример

Если $T = a^{2^n}, P = a^{2^m}$, то алгоритм Лившица будет работать за время $O(n^2 m)$ (при условии, что для этих текстов уже построены ПП), в то время как классический алгоритм КМП – за время $O(2^n + 2^m)$.

Теорема

Пусть $\mathcal{T}$ – ПП размера n, выводящая строку T, $\mathcal{P}$ – ПП размера m, выводящая строку P.
Алгоритм Лившица позволяет найти все вхождения текста P в текст T за время $O(n^2 m)$.

Пример

Если $T = a^{2^n}, P = a^{2^m}$, то алгоритм Лившица будет работать за время $O(n^2 m)$ (при условии, что для этих текстов уже построены ПП), в то время как классический алгоритм КМП – за время $O(2^n + 2^m)$.

Преимущества ПП

- Как правило, размер ПП меньше размера породившего ее текста
- Хорошо структурированы
- Полиномиально разрешимые классические строковые задачи, например:
  - Поиск сжатого образца в сжатом тексте
  - Поиск наибольшей общей подстроки двух сжатых строк
  - Поиск палиндромов в сжатой строке
  - Поиск квадратов в сжатой строке

Преимущества ПП

- Как правило, размер ПП меньше размера породившего ее текста
- Хорошо структурированы
- Полиномиально разрешимые классические строковые задачи, например:
  - Поиск сжатого образца в сжатом тексте
  - Поиск наибольшей общей подстроки двух сжатых строк
  - Поиск палиндромов в сжатой строке
  - Поиск квадратов в сжатой строке

Преимущества ПП

- Как правило, размер ПП меньше размера породившего ее текста
- Хорошо структурированы
- Полиномиально разрешимые классические строковые задачи, например:
  - Поиск сжатого образца в сжатом тексте
  - Поиск наибольшей общей подстроки двух сжатых строк
  - Поиск палиндромов в сжатой строке
  - Поиск квадратов в сжатой строке

Преимущества ПП

- Как правило, размер ПП меньше размера породившего ее текста
- Хорошо структурированы
- Полиномиально разрешимые классические строковые задачи, например:
  - Поиск сжатого образца в сжатом тексте
  - Поиск наибольшей общей подстроки двух сжатых строк
  - Поиск палиндромов в сжатой строке
  - Поиск квадратов в сжатой строке

Утверждение

Задача построения минимальной ПП, выводящей заданную строку T – NP-полная.

$\Downarrow$

Для построения ПП требуется использовать приближенные алгоритмы.

Утверждение

Задача построения минимальной ПП, выводящей заданную строку T – NP-полная.

$$\Downarrow$$

Для построения ПП требуется использовать приближенные алгоритмы.

Идея

Строить ПП на основе классических алгоритмов сжатия.

Алгоритм Лемпеля-Зива

ВХОД: T – строка над некоторым конечным алфавитом Σ
ВЫХОД: Набор объектов $m_1, m_2, \cdots, m_k$, где m_i является либо символом из алфавита Σ, либо парой чисел $(l, r), l < r$.
АЛГОРИТМ:

- $m_1 = T[1]$
- Пусть алгоритм отработал для префикса строки длины j и построил объекты $m_1, m_2, \cdots, m_i$. Тогда алгоритм ищет наибольший префикс $T[j+1 \cdots |T|]$, входящий как подстрока в $T[1 \cdots j]$. Если такой префикс есть и совпадает с подстрокой $T[l \cdots r]$, то $m_{i+1} = (l, r)$. Иначе, $m_{i+1} = T[j+1]$.

Идея

Строить ПП на основе классических алгоритмов сжатия.

Алгоритм Лемпеля-Зива

ВХОД: T – строка над некоторым конечным алфавитом Σ
ВЫХОД: Набор объектов $m_1, m_2, \cdots, m_k$, где m_i является либо символом из алфавита Σ, либо парой чисел $(l, r), l < r$.
АЛГОРИТМ:

- $m_1 = T[1]$
- Пусть алгоритм отработал для префикса строки длины j и построил объекты $m_1, m_2, \cdots, m_i$. Тогда алгоритм ищет наибольший префикс $T[j+1 \cdots |T|]$, входящий как подстрока в $T[1 \cdots j]$. Если такой префикс есть и совпадает с подстрокой $T[l \cdots r]$, то $m_{i+1} = (l, r)$. Иначе, $m_{i+1} = T[j+1]$.

Пример

Пусть T – строка «*abaababaabaab*».

- $m_1 = a$
- $m_2 = b$
- $m_3 = (1, 1)$
- $m_4 = (1, 3)$
- $m_5 = (2, 6)$
- $m_6 = (1, 2)$

Определение

LZ77-факторизация строки T – это набор слов $f_1, f_2, \cdots, f_k$, который строится следующим образом:

- $f_1 = T[1]$
- Пусть $T[1 \cdots j] = f_1 \cdot f_2 \cdot \ldots \cdot f_{i-1}$. Тогда f_i – наибольший префикс $T[j+1 \cdots |T|]$, входящий как подстрока в $T[1 \cdots j]$. Если такого префикса нет, то $f_i = T[j+1]$.

Пример

LZ77-факторизация слова «*abaababaabaab*» имеет вид:

$$a \cdot b \cdot a \cdot aba \cdot baaba \cdot ab.$$

Определение

Бинарное дерево называется *AVL-сбалансированным*, если для каждого его узла высоты левого и правого поддеревьев различаются не более, чем на 1.

Определение

Прямолинейную программу T будем называть *AVL-сбалансированной*, если ее дерево вывода является *AVL*-сбалансированным.

Определение

Бинарное дерево называется *AVL-сбалансированным*, если для каждого его узла высоты левого и правого поддеревьев различаются не более, чем на 1.

Определение

Прямолинейную программу T будем называть *AVL-сбалансированной*, если ее дерево вывода является *AVL*-сбалансированным.

Лемма

AVL-сбалансированное дерево из n узлов имеет высоту порядка $O(\log n)$.

Следствие

AVL-сбалансированная ПП T имеет высоту порядка $O(\log |T|)$.

Лемма

AVL-сбалансированное дерево из n узлов имеет высоту порядка $O(\log n)$.

Следствие

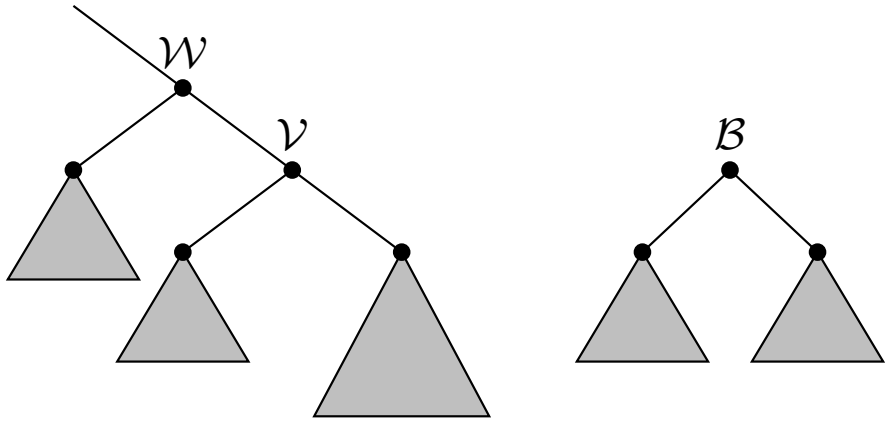
AVL-сбалансированная ПП T имеет высоту порядка $O(\log |T|)$.

Операция конкатенации двух *AVL-сбалансированных* ПП

Вход: $\mathcal{A}$, $\mathcal{B}$ – *AVL*-сбалансированные ПП.
Выход: *AVL*-сбалансированная ПП $\mathcal{G}$, выводящая $A \cdot B$.

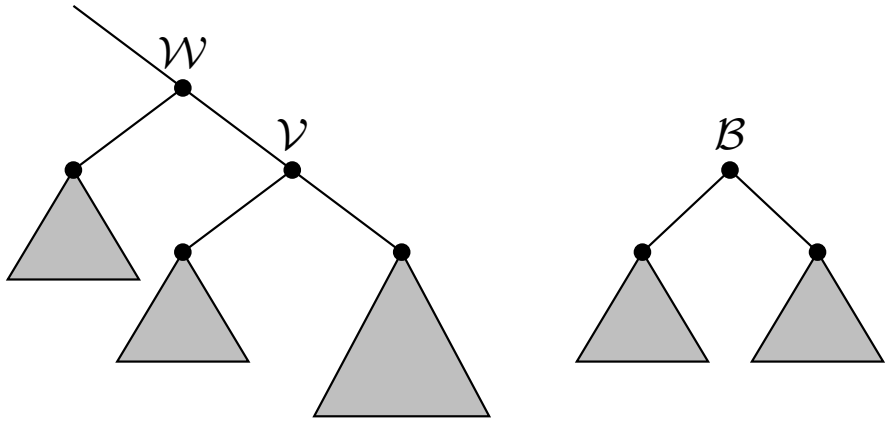
Алгоритм:

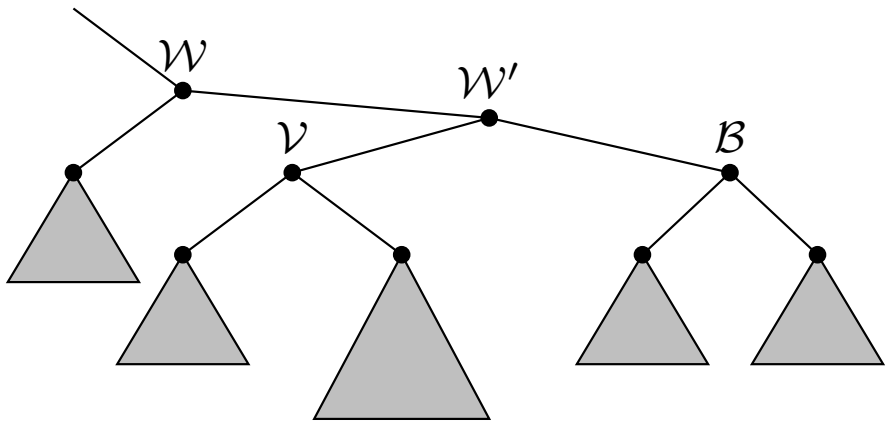
Алгоритм:



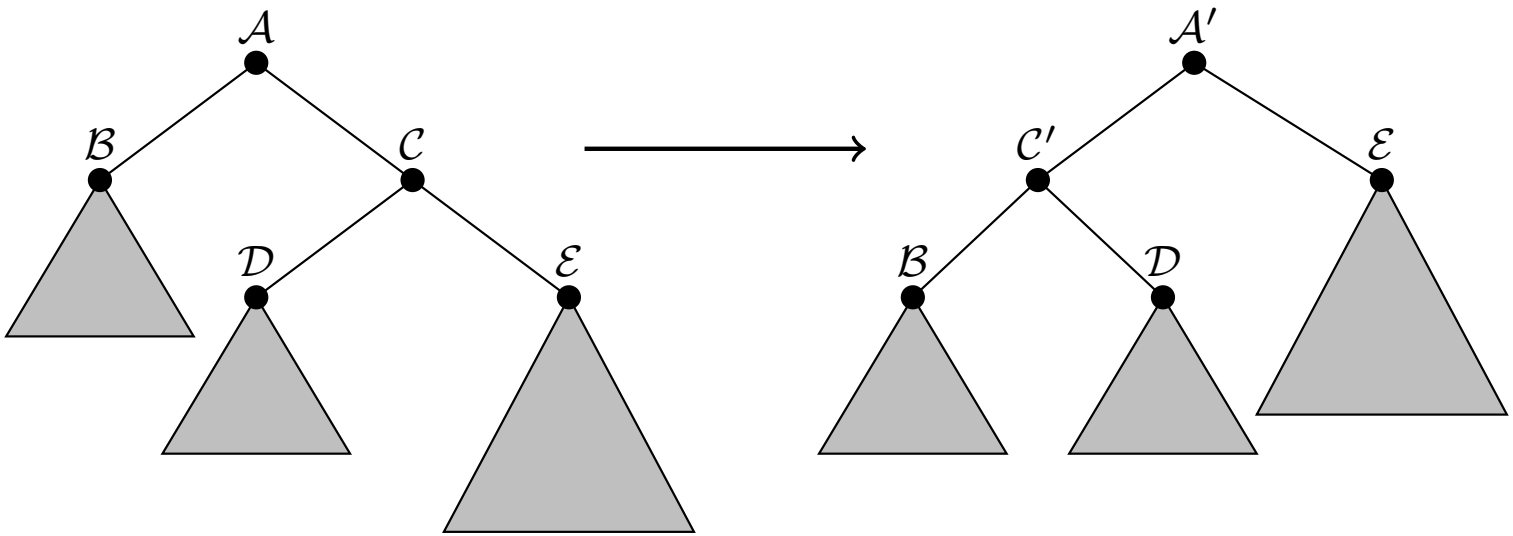
-

Алгоритм:


$\mathcal{W}$
$\mathcal{V}$
$\mathcal{B}$


$\mathcal{W}$
$\mathcal{W}'$
$\mathcal{V}$
$\mathcal{B}$

Операция перебалансировки:



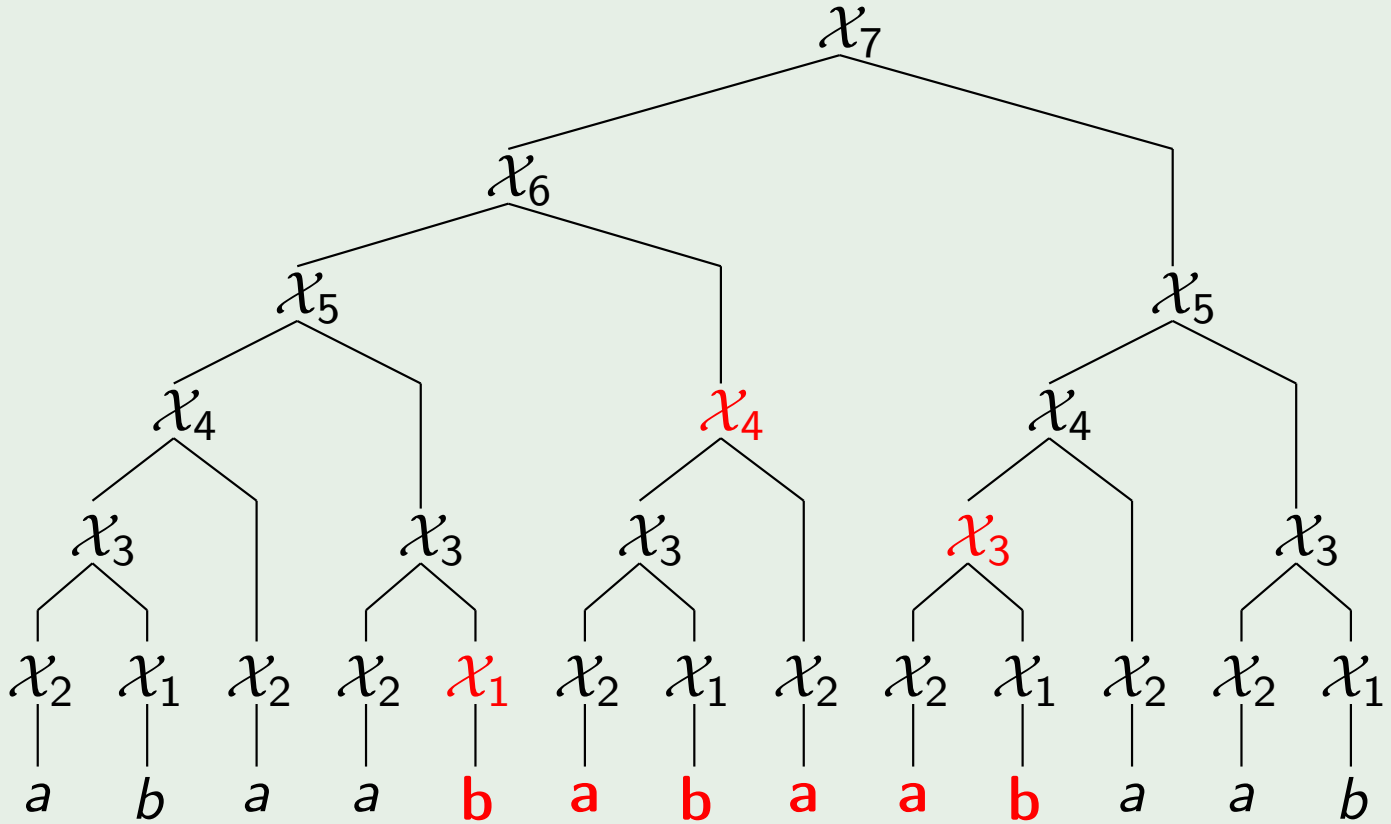
Асмптотическая сложность операции конкатенации

Операция конкатенации двух *AVL*-сбалансированных ПП $\mathcal{A}$ и $\mathcal{B}$ выполняется за время $O(h(Tree(\mathcal{A})) - h(Tree(\mathcal{B})))$, где $Tree(\mathcal{X}))$ – дерево вывода ПП $\mathcal{X}$. Также в ходе операции порождается $O(h(Tree(\mathcal{A})) - h(Tree(\mathcal{B})))$ новых правил вывода.

Операция взятия подстроки

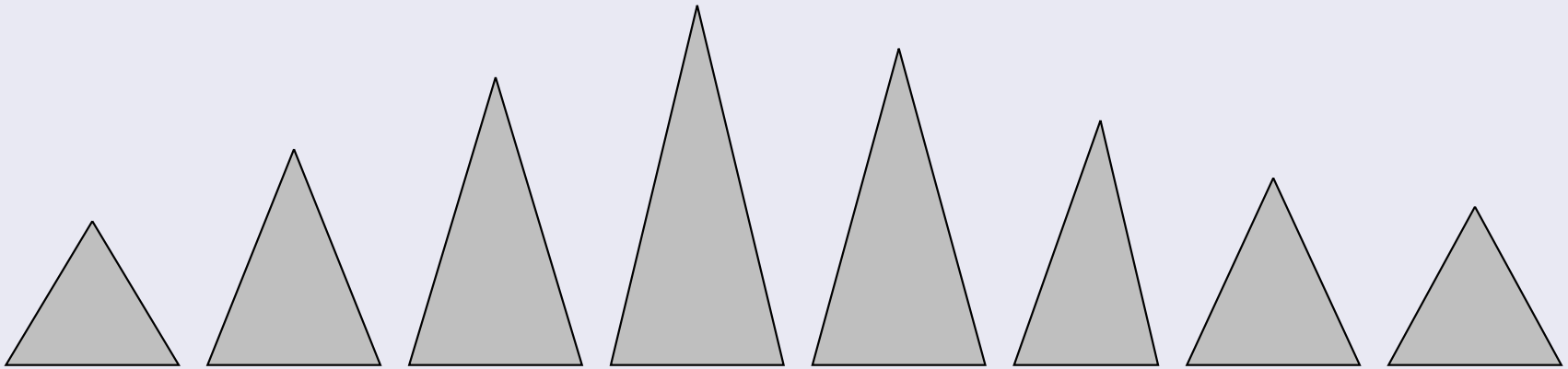
Вход: $\mathcal{T}$ – *AVL*-сбалансированная ПП, выводящая строку T и целые числа l и r, такие, что $1 \leq l < r \leq |T|$.
Выход: *AVL*-сбалансированная ПП $\mathcal{S}$, выводящая строку $T[l \dots r]$.

Выбираем правила, составляющие подстроку



Утверждение

Выбранный набор правил всегда имеет вид:



Асмптотическая сложность операции взятия подстроки

Для *AVL*-сбалансированной ПП $\mathcal{T}$, выводящей строку T, операция взятия подстроки выполняется за время $O(\log|T|)$, при этом порождает не более $O(\log|T|)$ новых правил вывода.

Алгоритм Риттера

Вход: Строка T длины n, факторизация $F(T) : T = f_1 \cdot f_2 \cdot \ldots f_k$. Также для каждого фактора f_i, если он не однобуквенный, известна позиция его вхождения в строку $T_{i-1} = f_1 \cdot f_2 \cdot \ldots \cdot f_{i-1}$, то есть, известны числа l_i и t_i такие, что $f_i = T[l_i \ldots r_i]$.

Выход: ПП $\mathcal{T}$, выводящая T.

Ход алгоритма:

- Строим тривиальную *AVL*-сбалансированную ПП $\mathcal{T}_1$, выводящую однобуквенный фактор f_1.
- Возьмем фактор f_{i+1}. Пусть $f_{i+1} = T_i[l_i \dots r_i]$.
- С помощью операции взятия подстроки построим ПП $\mathcal{S}$, выводящую строку $T_i[l_i \dots r_i]$.
- С помощью алгоритма конкатенации построим ПП $\mathcal{T}_{i+1} = \mathcal{T}_i \cdot \mathcal{S}$.

Теорема

Пусть дан текст T и его *LZ77*-факторизация размера k. Тогда **алгоритм Риттера** позволяет построить ПП $\mathcal{T}$ размером $O(k \cdot \log |T|)$ за время $O(k \cdot \log |T|)$.

Узкое место алгоритма Риттера

$S = ba^{2^n} ba^{2^{n-1}} \dots ba.$

LZ77-факторизация этой строки:

$b \cdot a \cdot a \cdot a^2 \cdot a^4 \cdot \dots \cdot a^{2^{n-1}-1} \cdot ba^{2^{n-1}} \cdot ba^{2^{n-2}} \cdot \dots \cdot ba.$

Число операций перебалансировки:

$\sum_{i=0}^{n-1}(n + (n-1-i) - (i)) = \sum_{i=0}^{n-1}(2n - 2i - 1) = n^2$

Функция минимального количества перебалансировок

Пусть есть набор правил $\mathcal{F}_1, \mathcal{F}_2, \ldots \mathcal{F}_k$.

$$\varphi(i,j) = \begin{cases} 0, \text{если } i = j \\ \min_{r=i}^{j}(\varphi(i,r) + \varphi(r+1,j) + |\log(|F_i| + \cdots + |F_r|) - \\ - \log(|F_{r+1}| + \cdots + |F_j|)|), \text{иначе.} \end{cases}$$

Утверждение

Функция $\varphi(\cdot,\cdot)$ может быть вычислена за время $O(k^3)$.

Функция минимального количества перебалансировок

Пусть есть набор правил $\mathcal{F}_1, \mathcal{F}_2, \ldots \mathcal{F}_k$.

$$\varphi(i,j) = \begin{cases} 0, \text{если } i = j \\ \min_{r=i}^{j}(\varphi(i,r) + \varphi(r+1,j) + |\log(|F_i| + \cdots + |F_r|) - \\ - \log(|F_{r+1}| + \cdots + |F_j|)|), \text{иначе.} \end{cases}$$

Утверждение

Функция $\varphi(\cdot,\cdot)$ может быть вычислена за время $O(k^3)$.

Модернизированный алгоритм Риттера

Вход: Строка T длины n, факторизация $F(T) : T = f_1 \cdot f_2 \cdot \ldots f_k$. Также для каждого фактора f_i, если он не однобуквенный, известна позиция его вхождения в строку $T_{i-1} = f_1 \cdot f_2 \cdot \ldots \cdot f_{i-1}$, то есть, известны числа l_i и t_i такие, что $f_i = T[l_i \ldots r_i]$.

Выход: ПП $\mathcal{T}$, выводящая T.

Алгоритм:

База: Построим тривиальную *AVL*-сбалансированную ПП $\mathcal{T}_1$ для однобуквенного фактора f_1.

Шаг: Пусть мы построили *AVL*-сбалансированную ПП $\mathcal{T}_i$ для строки $T_i = f_1 \cdot f_2 \cdot \ldots \cdot f_i$.

1. Возьмем факторы $f_{i+1}, f_{i+1}, \ldots, f_{i+r}$, такие что, каждый из них входит как подстрока в T_i, а f_{i+r+1} уже не входит.
2. Для каждого фактора f_{i+s} с помощью операции взятия подстроки построим ПП $\mathcal{F}_s$, выводящую f_{i+s}.
3. Для полученного набора правил вычислим таблицу $\varphi(i, j)$.
4. Применим к набору правил алгоритм конкатенации в порядке, который диктует функция $\varphi(i, j)$. Таким образом, построим ПП $\mathcal{S}$, выводящую $F_1 \cdot F_2 \cdot \ldots F_r$
5. С помощью алгоритма конкатенации построим ПП $\mathcal{T}_{i+r} = \mathcal{T}_i \cdot \mathcal{S}$.

Алгоритм:

База: Построим тривиальную *AVL*-сбалансированную ПП $\mathcal{T}_1$ для однобуквенного фактора f_1.

Шаг: Пусть мы построили *AVL*-сбалансированную ПП $\mathcal{T}_i$ для строки $T_i = f_1 \cdot f_2 \cdot \ldots \cdot f_i$.

1. Возьмем факторы $f_{i+1}, f_{i+1}, \ldots, f_{i+r}$, такие что, каждый из них входит как подстрока в T_i, а f_{i+r+1} уже не входит.
2. Для каждого фактора f_{i+s} с помощью операции взятия подстроки построим ПП $\mathcal{F}_s$, выводящую f_{i+s}.
3. Для полученного набора правил вычислим таблицу $\varphi(i, j)$.
4. Применим к набору правил алгоритм конкатенации в порядке, который диктует функция $\varphi(i, j)$. Таким образом, построим ПП $\mathcal{S}$, выводящую $F_1 \cdot F_2 \cdot \ldots F_r$
5. С помощью алгоритма конкатенации построим ПП $\mathcal{T}_{i+r} = \mathcal{T}_i \cdot \mathcal{S}$.

Алгоритм:

База: Построим тривиальную *AVL*-сбалансированную ПП $\mathcal{T}_1$ для однобуквенного фактора f_1.

Шаг: Пусть мы построили *AVL*-сбалансированную ПП $\mathcal{T}_i$ для строки $T_i = f_1 \cdot f_2 \cdot \ldots \cdot f_i$.

1. Возьмем факторы $f_{i+1}, f_{i+1}, \ldots, f_{i+r}$, такие что, каждый из них входит как подстрока в T_i, а f_{i+r+1} уже не входит.
2. Для каждого фактора f_{i+s} с помощью операции взятия подстроки построим ПП $\mathcal{F}_s$, выводящую f_{i+s}.
3. Для полученного набора правил вычислим таблицу $\varphi(i, j)$.
4. Применим к набору правил алгоритм конкатенации в порядке, который диктует функция $\varphi(i, j)$. Таким образом, построим ПП $\mathcal{S}$, выводящую $F_1 \cdot F_2 \cdot \ldots F_r$
5. С помощью алгоритма конкатенации построим ПП $\mathcal{T}_{i+r} = \mathcal{T}_i \cdot \mathcal{S}$.

Алгоритм:

База: Построим тривиальную *AVL*-сбалансированную ПП $\mathcal{T}_1$ для однобуквенного фактора f_1.

Шаг: Пусть мы построили *AVL*-сбалансированную ПП $\mathcal{T}_i$ для строки $T_i = f_1 \cdot f_2 \cdot \ldots \cdot f_i$.

1. Возьмем факторы $f_{i+1}, f_{i+1}, \ldots, f_{i+r}$, такие что, каждый из них входит как подстрока в T_i, а f_{i+r+1} уже не входит.
2. Для каждого фактора f_{i+s} с помощью операции взятия подстроки построим ПП $\mathcal{F}_s$, выводящую f_{i+s}.
3. Для полученного набора правил вычислим таблицу $\varphi(i, j)$.
4. Применим к набору правил алгоритм конкатенации в порядке, который диктует функция $\varphi(i, j)$. Таким образом, построим ПП $\mathcal{S}$, выводящую $F_1 \cdot F_2 \cdot \ldots F_r$
5. С помощью алгоритма конкатенации построим ПП $\mathcal{T}_{i+r} = \mathcal{T}_i \cdot \mathcal{S}$.

Алгоритм:

База: Построим тривиальную *AVL*-сбалансированную ПП $\mathcal{T}_1$ для однобуквенного фактора f_1.

Шаг: Пусть мы построили *AVL*-сбалансированную ПП $\mathcal{T}_i$ для строки $T_i = f_1 \cdot f_2 \cdot \ldots \cdot f_i$.

1. Возьмем факторы $f_{i+1}, f_{i+1}, \ldots, f_{i+r}$, такие что, каждый из них входит как подстрока в T_i, а f_{i+r+1} уже не входит.
2. Для каждого фактора f_{i+s} с помощью операции взятия подстроки построим ПП $\mathcal{F}_s$, выводящую f_{i+s}.
3. Для полученного набора правил вычислим таблицу $\varphi(i, j)$.
4. Применим к набору правил алгоритм конкатенации в порядке, который диктует функция $\varphi(i, j)$. Таким образом, построим ПП $\mathcal{S}$, выводящую $F_1 \cdot F_2 \cdot \ldots F_r$
5. С помощью алгоритма конкатенации построим ПП $\mathcal{T}_{i+r} = \mathcal{T}_i \cdot \mathcal{S}$.

Алгоритм:

База: Построим тривиальную *AVL*-сбалансированную ПП $\mathcal{T}_1$ для однобуквенного фактора f_1.

Шаг: Пусть мы построили *AVL*-сбалансированную ПП $\mathcal{T}_i$ для строки $T_i = f_1 \cdot f_2 \cdot \ldots \cdot f_i$.

1. Возьмем факторы $f_{i+1}, f_{i+1}, \ldots, f_{i+r}$, такие что, каждый из них входит как подстрока в T_i, а f_{i+r+1} уже не входит.
2. Для каждого фактора f_{i+s} с помощью операции взятия подстроки построим ПП $\mathcal{F}_s$, выводящую f_{i+s}.
3. Для полученного набора правил вычислим таблицу $\varphi(i, j)$.
4. Применим к набору правил алгоритм конкатенации в порядке, который диктует функция $\varphi(i, j)$. Таким образом, построим ПП $\mathcal{S}$, выводящую $F_1 \cdot F_2 \cdot \ldots F_r$
5. С помощью алгоритма конкатенации построим ПП $\mathcal{T}_{i+r} = \mathcal{T}_i \cdot \mathcal{S}$.

Алгоритм:

База: Построим тривиальную *AVL*-сбалансированную ПП $\mathcal{T}_1$ для однобуквенного фактора f_1.

Шаг: Пусть мы построили *AVL*-сбалансированную ПП $\mathcal{T}_i$ для строки $T_i = f_1 \cdot f_2 \cdot \ldots \cdot f_i$.

1. Возьмем факторы $f_{i+1}, f_{i+1}, \ldots, f_{i+r}$, такие что, каждый из них входит как подстрока в T_i, а f_{i+r+1} уже не входит.
2. Для каждого фактора f_{i+s} с помощью операции взятия подстроки построим ПП $\mathcal{F}_s$, выводящую f_{i+s}.
3. Для полученного набора правил вычислим таблицу $\varphi(i, j)$.
4. Применим к набору правил алгоритм конкатенации в порядке, который диктует функция $\varphi(i, j)$. Таким образом, построим ПП $\mathcal{S}$, выводящую $F_1 \cdot F_2 \cdot \ldots F_r$
5. С помощью алгоритма конкатенации построим ПП $\mathcal{T}_{i+r} = \mathcal{T}_i \cdot \mathcal{S}$.

Алгоритм:

База: Построим тривиальную *AVL*-сбалансированную ПП $\mathcal{T}_1$ для однобуквенного фактора f_1.

Шаг: Пусть мы построили *AVL*-сбалансированную ПП $\mathcal{T}_i$ для строки $T_i = f_1 \cdot f_2 \cdot \ldots \cdot f_i$.

1. Возьмем факторы $f_{i+1}, f_{i+1}, \ldots, f_{i+r}$, такие что, каждый из них входит как подстрока в T_i, а f_{i+r+1} уже не входит.
2. Для каждого фактора f_{i+s} с помощью операции взятия подстроки построим ПП $\mathcal{F}_s$, выводящую f_{i+s}.
3. Для полученного набора правил вычислим таблицу $\varphi(i, j)$.
4. Применим к набору правил алгоритм конкатенации в порядке, который диктует функция $\varphi(i, j)$. Таким образом, построим ПП $\mathcal{S}$, выводящую $F_1 \cdot F_2 \cdot \ldots F_r$
5. С помощью алгоритма конкатенации построим ПП $\mathcal{T}_{i+r} = \mathcal{T}_i \cdot \mathcal{S}$.

Теорема

Пусть дан текст T и его *LZ77*-факторизация размера k. Тогда **модернизированный алгоритм Риттера** позволяет построить ПП $\mathcal{T}$ размером $O(k \cdot \log |T|)$ за время $O(k \cdot \log |T|)$.

Число перебалансировок