Прямолинейные программы: практический тест

Иван Бурмистров

8 июня 2011

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.
- Сокращение размера входа 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.
- Сокращение размера входа 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.
- Сокращение размера входа за счет предварительной обработки.

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.

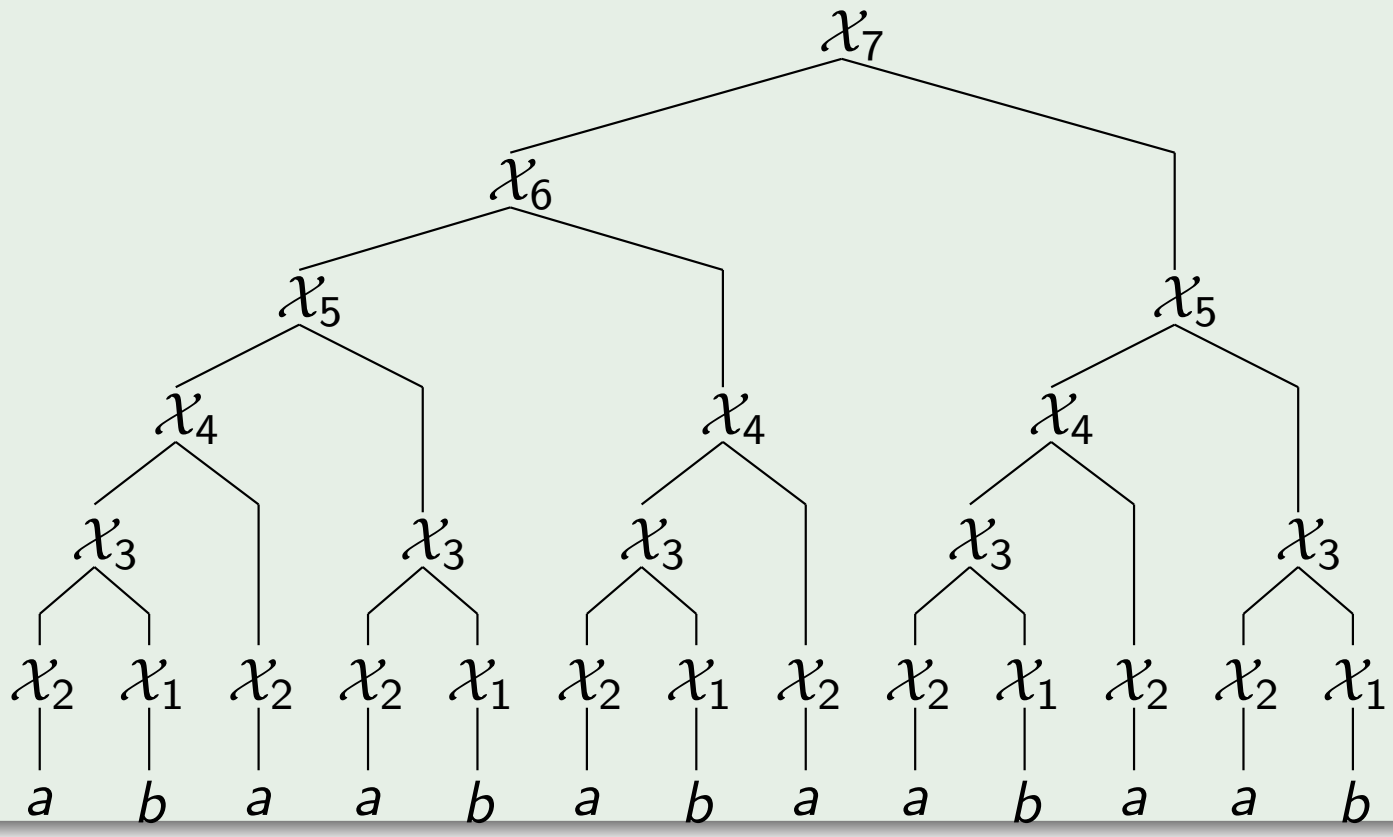
Пример

Рассмотрим ПП $\mathcal{X}$, выводящую строку $X = «abaababaabaab»$.

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

Пример

Графическое изображение ПП:



Существует ряд классических задач, сформулированных в терминах ПП и разрешимых за полиномиальное время от размера ПП, например:

- Поиск сжатого образца в сжатом тексте
- Поиск наибольшей общей подстроки двух сжатых строк
- Поиск палиндромов в сжатой строке
- Поиск квадратов в сжатой строке

Существует ряд классических задач, сформулированных в терминах ПП и разрешимых за полиномиальное время от размера ПП, например:

- Поиск сжатого образца в сжатом тексте
- Поиск наибольшей общей подстроки двух сжатых строк
- Поиск палиндромов в сжатой строке
- Поиск квадратов в сжатой строке

Существует ряд классических задач, сформулированных в терминах ПП и разрешимых за полиномиальное время от размера ПП, например:

- Поиск сжатого образца в сжатом тексте
- Поиск наибольшей общей подстроки двух сжатых строк
- Поиск палиндромов в сжатой строке
- Поиск квадратов в сжатой строке

Существует ряд классических задач, сформулированных в терминах ПП и разрешимых за полиномиальное время от размера ПП, например:

- Поиск сжатого образца в сжатом тексте
- Поиск наибольшей общей подстроки двух сжатых строк
- Поиск палиндромов в сжатой строке
- Поиск квадратов в сжатой строке

Существует ряд классических задач, сформулированных в терминах ПП и разрешимых за полиномиальное время от размера ПП, например:

- Поиск сжатого образца в сжатом тексте
- Поиск наибольшей общей подстроки двух сжатых строк
- Поиск палиндромов в сжатой строке
- Поиск квадратов в сжатой строке

Пример

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 $O(2^n + 2^m)$.

Пример

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 $O(2^n + 2^m)$.

Утверждение

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

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

Для построения ПП в качестве отправной точки использовать классические алгоритмы сжатия.

Оказалось, что классический алгоритм Лемпеля-Зива хорошо подходит для этой задачи.

Идея

Для построения ПП в качестве отправной точки использовать классические алгоритмы сжатия.

Оказалось, что классический **алгоритм Лемпеля-Зива** хорошо подходит для этой задачи.

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 b a aba baaba ab.

LZ77-факторизацию строки T длины n можно построить как минимум двумя способами:

- За $O(n)$ с использованием **суффиксного дерева**. Тяжело адаптируется на случай, когда заканчивается оперативная память.
- За $O(n \log n)$ с использованием **суффиксного массива**. Проще в реализации, легко адаптируется на использование внешней памяти.

LZ77-факторизацию строки T длины n можно построить как минимум двумя способами:

- За $O(n)$ с использованием **суффиксного дерева**. Тяжело адаптируется на случай, когда заканчивается оперативная память.
- За $O(n \log n)$ с использованием **суффиксного массива**. Проще в реализации, легко адаптируется на использование внешней памяти.

LZ77-факторизацию строки T длины n можно построить как минимум двумя способами:

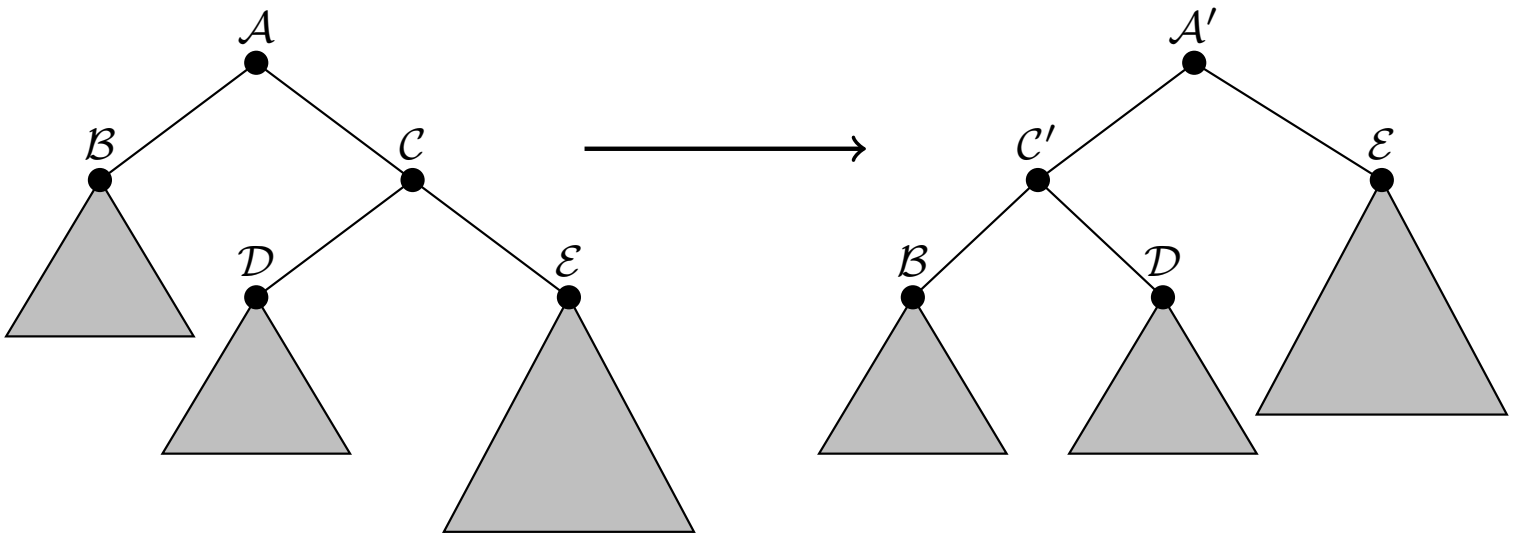
- За $O(n)$ с использованием **суффиксного дерева**. Тяжело адаптируется на случай, когда заканчивается оперативная память.
- За $O(n \log n)$ с использованием **суффиксного массива**. Проще в реализации, легко адаптируется на использование внешней памяти.

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 $O(k \cdot \log |T|)$.

Основной объект в алгоритме Риттера – **сбалансированное бинарное дерево**.

А одна из основных операций – **перебалансировка** узла:



Мысль

Когда для хранения дерева не хватает оперативной памяти, перебалансировка узла становится **дорогой операцией**.

⇓

Попробуем модернизировать алгоритм с целью уменьшения числа перебалансировок.

Мысль

Когда для хранения дерева не хватает оперативной памяти, перебалансировка узла становится **дорогой операцией**.

$$\Downarrow$$

Попробуем модернизировать алгоритм с целью уменьшения числа перебалансировок.

Результат

Удалось модернизировать алгоритм Риттера таким образом, чтобы он совершал существенно меньшее число перебалансировок и на практике работал быстрее.

Практические результаты

Природа текстов влияет на время обработки текстов и показатели сжатия. В данной работе были рассмотрены следующие типы текстов:

- **ДНК** – строки, которые используются на практике
- **случайные строки над 4-буквенным алфавитом** – предположительно худший вход для алгоритмов сжатия
- **строки Фибоначчи** – предположительно один из лучших входов для алгоритмов сжатия

Практические результаты

Природа текстов влияет на время обработки текстов и показатели сжатия. В данной работе были рассмотрены следующие типы текстов:

- **ДНК** – строки, которые используются на практике
- **случайные строки над 4-буквенным алфавитом** – предположительно худший вход для алгоритмов сжатия
- **строки Фибоначчи** – предположительно один из лучших входов для алгоритмов сжатия

Практические результаты

Природа текстов влияет на время обработки текстов и показатели сжатия. В данной работе были рассмотрены следующие типы текстов:

- **ДНК** – строки, которые используются на практике
- **случайные строки над 4-буквенным алфавитом** – предположительно худший вход для алгоритмов сжатия
- **строки Фибоначчи** – предположительно один из лучших входов для алгоритмов сжатия

Практические результаты

Природа текстов влияет на время обработки текстов и показатели сжатия. В данной работе были рассмотрены следующие типы текстов:

- **ДНК** – строки, которые используются на практике
- **случайные строки над 4-буквенным алфавитом** – предположительно худший вход для алгоритмов сжатия
- **строки Фибоначчи** – предположительно один из лучших входов для алгоритмов сжатия

Будут приведены результаты тестирования следующих алгоритмов:

- Алгоритм *LZ77*-факторизации. Обозначение –**lz77**
- Алгоритм Риттера. Обозначение – **SLPClassic**
- Модернизированный алгоритм Риттера. Обозначение – **SLPnew**

Будут приведены результаты тестирования следующих алгоритмов:

- Алгоритм *LZ77*-факторизации. Обозначение –**lz77**
- Алгоритм Риттера. Обозначение – **SLPClassic**
- Модернизированный алгоритм Риттера. Обозначение – **SLPnew**

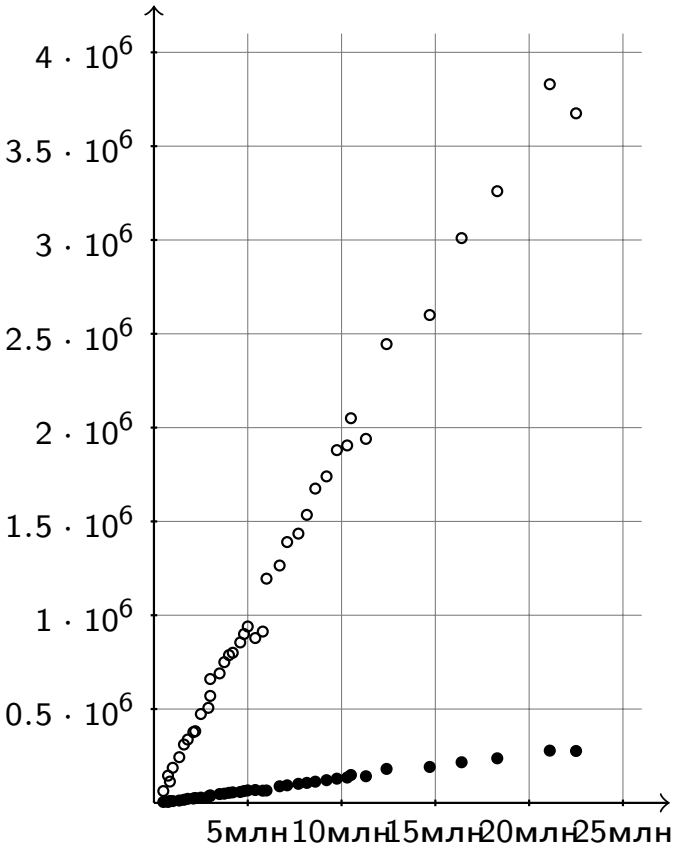
Будут приведены результаты тестирования следующих алгоритмов:

- Алгоритм *LZ77*-факторизации. Обозначение –**lz77**
- Алгоритм Риттера. Обозначение – **SLPClassic**
- Модернизированный алгоритм Риттера. Обозначение – **SLPnew**

Будут приведены результаты тестирования следующих алгоритмов:

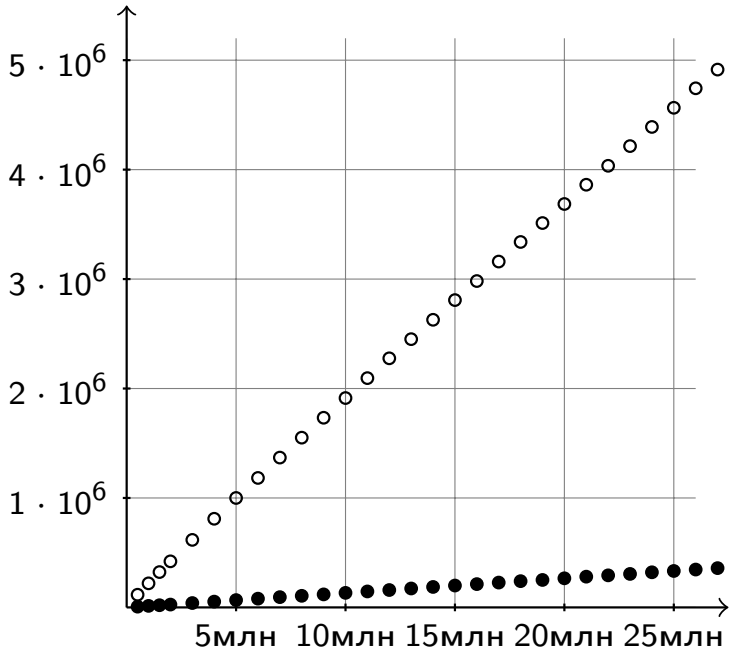
- Алгоритм *LZ77*-факторизации. Обозначение –**lz77**
- Алгоритм Риттера. Обозначение – **SLPClassic**
- Модернизированный алгоритм Риттера. Обозначение – **SLPnew**

Число перебалансировок



○ – SLPclassic • – SLPnew

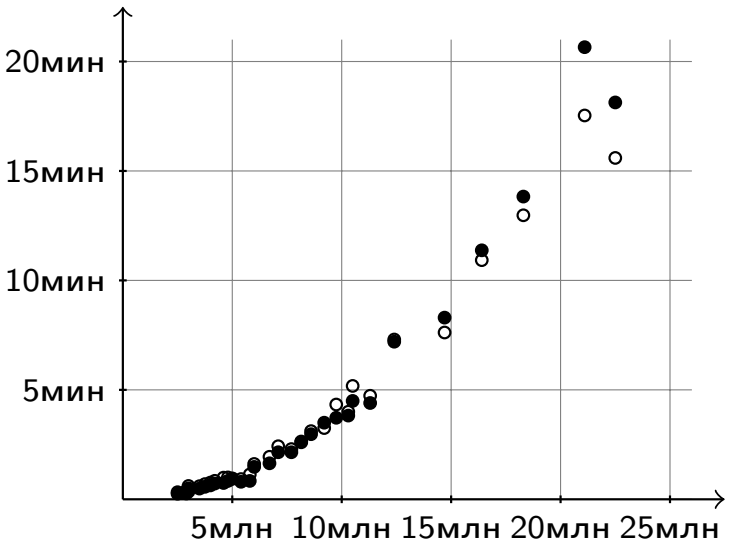
Строки ДНК



○ – SLPclassic • – SLPnew

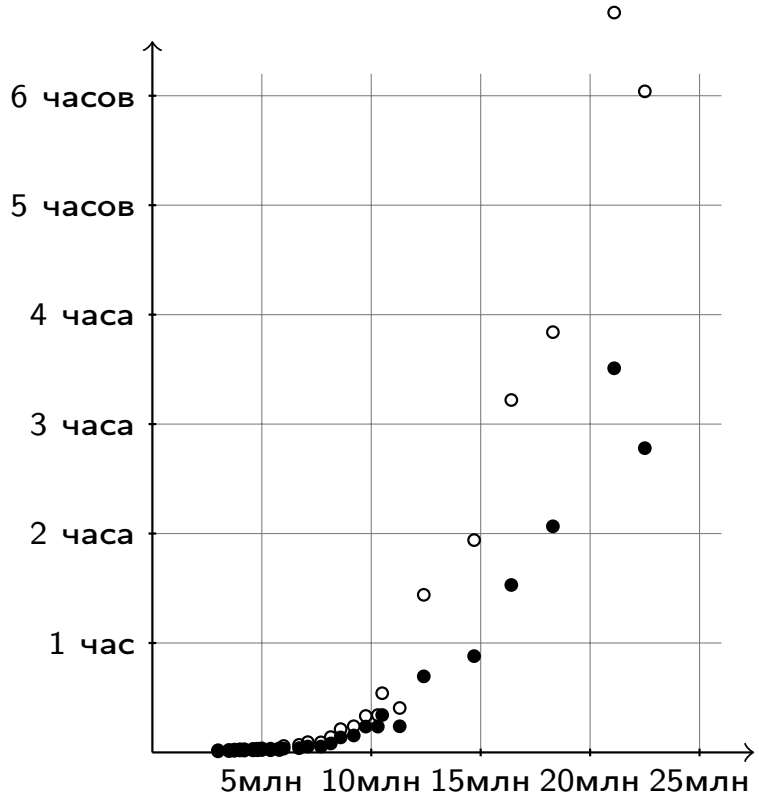
Случайные строки

Скорость работы на строках ДНК



∘ – SLPclassic • – SLPnew

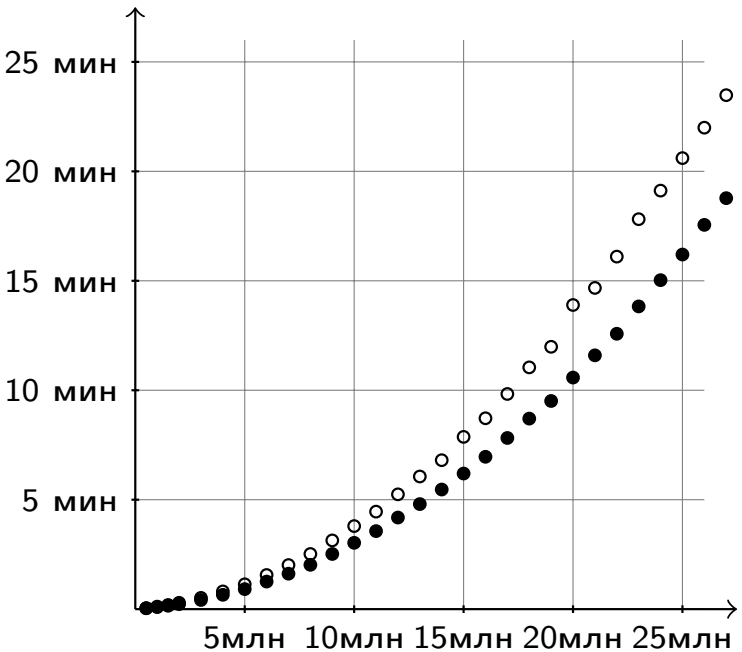
Оперативная память



∘ – SLPclassic • – SLPnew

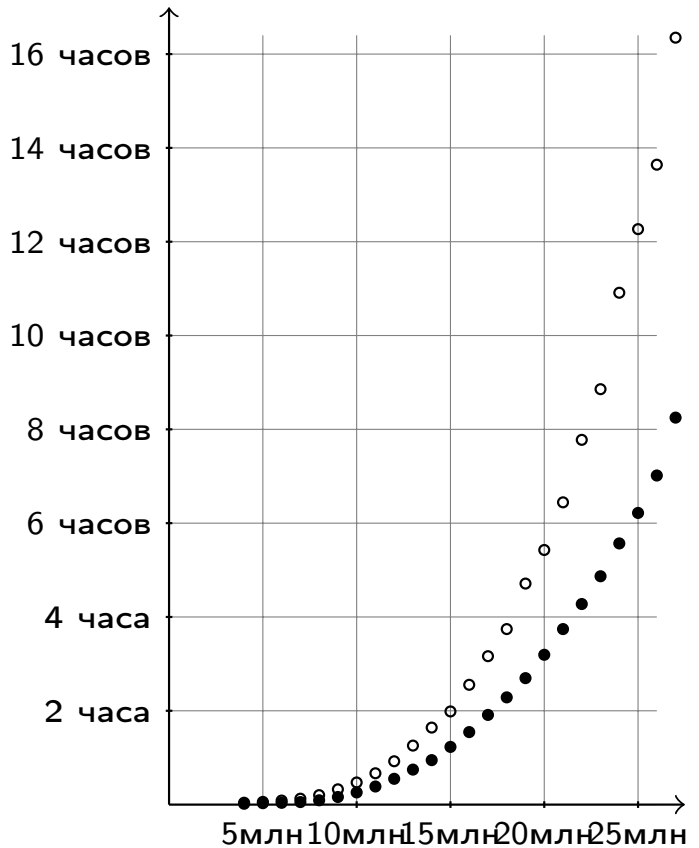
Внешняя память

Скорость работы на случайных строках



○ – **SLPclassic** • – **SLPnew**

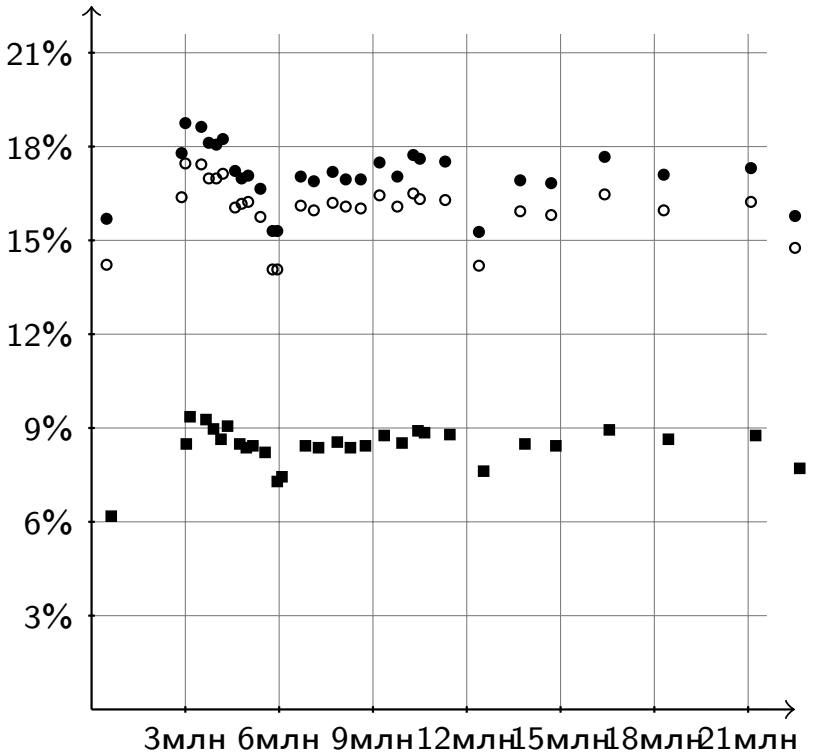
Оперативная память



○ – **SLPclassic** • – **SLPnew**

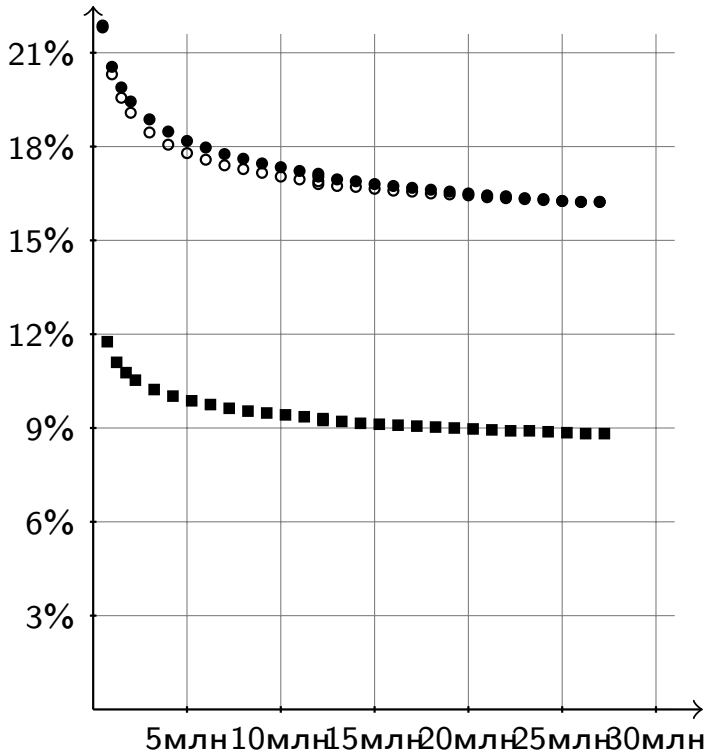
Внешняя память

Коэффициент сжатия



◦ – **SLPclassic** • – **SLPnew** ▪ – **lz77**

Строки ДНК



◦ – **SLPclassic** • – **SLPnew** ▪ – **lz77**

Случайные строки

Результаты

- Альтернативный алгоритм *LZ77*-факторизации
- Модернизированный алгоритм Риттера
- Тесты

Результаты

- Альтернативный алгоритм *LZ77*-факторизации
- Модернизированный алгоритм Риттера
- Тесты

Результаты

- Альтернативный алгоритм *LZ77*-факторизации
- Модернизированный алгоритм Риттера
- Тесты

Результаты

- Альтернативный алгоритм *LZ77*-факторизации
- Модернизированный алгоритм Риттера
- Тесты

Планы

- Довести альтернативный алгоритм *LZ77*-факторизации до асимптотики $O(n)$
- Алгоритм факторизации, эффективный с точки зрения операций чтения-записи
- Алгоритм построения ПП, эффективный с точки зрения операций чтения-записи
- Худший случай алгоритма Риттера или уточнение оценки размера ПП
- Классификация строк по размеру ПП
- Поиск сфер применения ПП (криптография?)

Планы

- Довести альтернативный алгоритм *LZ77*-факторизации до асимптотики $O(n)$
- Алгоритм факторизации, эффективный с точки зрения операций чтения-записи
- Алгоритм построения ПП, эффективный с точки зрения операций чтения-записи
- Худший случай алгоритма Риттера или уточнение оценки размера ПП
- Классификация строк по размеру ПП
- Поиск сфер применения ПП (криптография?)

Планы

- Довести альтернативный алгоритм *LZ77*-факторизации до асимптотики $O(n)$
- Алгоритм факторизации, эффективный с точки зрения операций чтения-записи
- Алгоритм построения ПП, эффективный с точки зрения операций чтения-записи
- Худший случай алгоритма Риттера или уточнение оценки размера ПП
- Классификация строк по размеру ПП
- Поиск сфер применения ПП (криптография?)

Планы

- Довести альтернативный алгоритм *LZ77*-факторизации до асимптотики $O(n)$
- Алгоритм факторизации, эффективный с точки зрения операций чтения-записи
- Алгоритм построения ПП, эффективный с точки зрения операций чтения-записи
- Худший случай алгоритма Риттера или уточнение оценки размера ПП
- Классификация строк по размеру ПП
- Поиск сфер применения ПП (криптография?)

Планы

- Довести альтернативный алгоритм *LZ77*-факторизации до асимптотики $O(n)$
- Алгоритм факторизации, эффективный с точки зрения операций чтения-записи
- Алгоритм построения ПП, эффективный с точки зрения операций чтения-записи
- Худший случай алгоритма Риттера или уточнение оценки размера ПП
- Классификация строк по размеру ПП
- Поиск сфер применения ПП (криптография?)

Планы

- Довести альтернативный алгоритм *LZ77*-факторизации до асимптотики $O(n)$
- Алгоритм факторизации, эффективный с точки зрения операций чтения-записи
- Алгоритм построения ПП, эффективный с точки зрения операций чтения-записи
- Худший случай алгоритма Риттера или уточнение оценки размера ПП
- Классификация строк по размеру ПП
- Поиск сфер применения ПП (криптография?)

Планы

- Довести альтернативный алгоритм *LZ77*-факторизации до асимптотики $O(n)$
- Алгоритм факторизации, эффективный с точки зрения операций чтения-записи
- Алгоритм построения ПП, эффективный с точки зрения операций чтения-записи
- Худший случай алгоритма Риттера или уточнение оценки размера ПП
- Классификация строк по размеру ПП
- Поиск сфер применения ПП (криптография?)