Исследование алгоритмов поиска на сжатом представлении данных

Плинер Юрий

11 июня 2014

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.

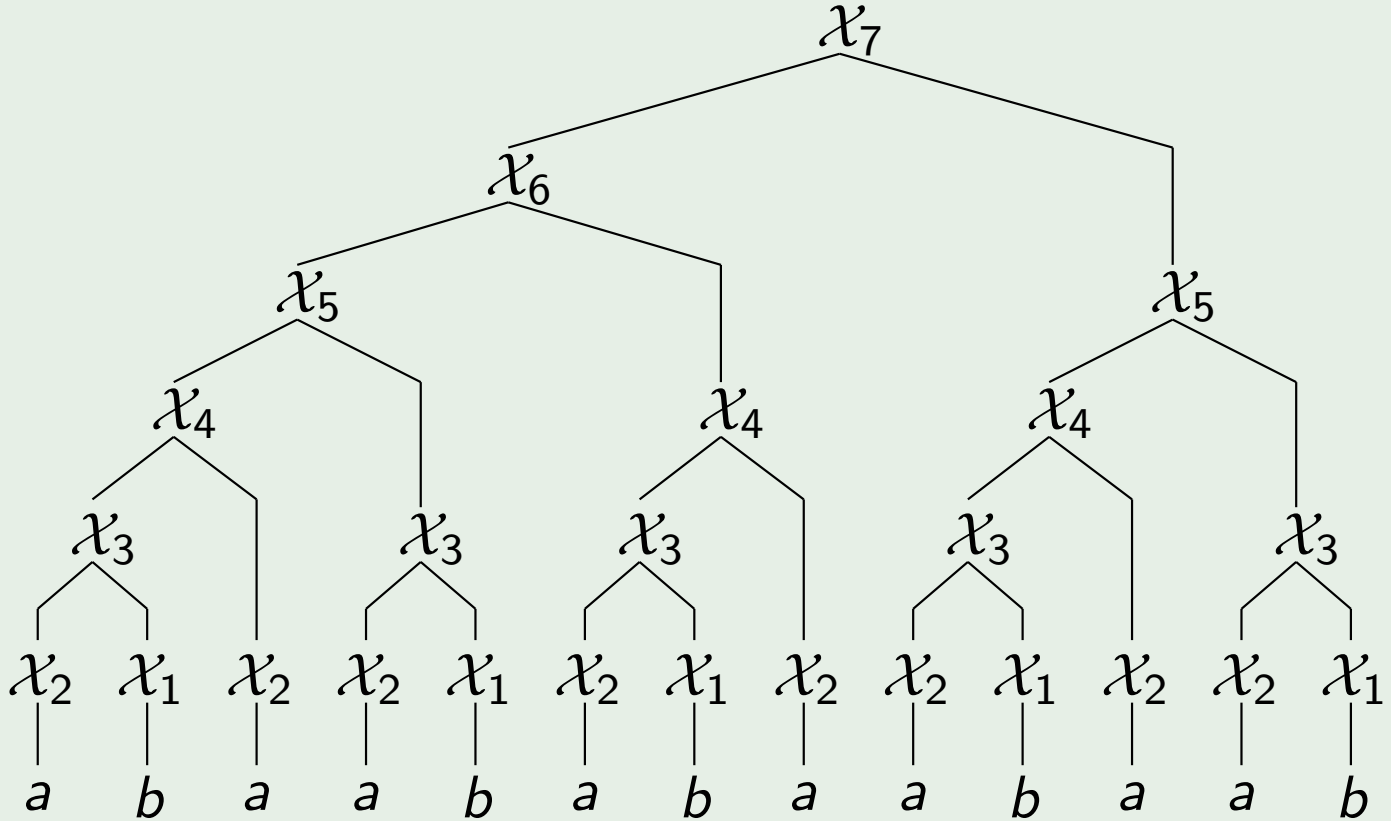
ПП $\mathcal{X}$, выводящая строку «*abaababaabaab*»

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



Преимущества ПП

- Хорошо структурированы
- Имеется связь с алгоритмами сжатия, использующимися на практике
- 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

- Хорошо структурированы
- Имеется связь с алгоритмами сжатия, использующимися на практике
- 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

- Хорошо структурированы
- Имеется связь с алгоритмами сжатия, использующимися на практике
- 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

- Хорошо структурированы
- Имеется связь с алгоритмами сжатия, использующимися на практике
- 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

На вход

ПП размером n выводящая текст и ПП размером m выводящая шаблон.

На выход

Позиции вхождения шаблона в текст.

Алгоритмическая сложность

Алгоритм работает за $O(n^2 \cdot m)$ по времени и за $O(n \cdot m)$ по памяти.

На вход

ПП размером n выводящая текст и ПП размером m выводящая шаблон.

На выход

Позиции вхождения шаблона в текст.

Алгоритмическая сложность

Алгоритм работает за $O(n^2 \cdot m)$ по времени и за $O(n \cdot m)$ по памяти.

На вход

ПП размером n выводящая текст и ПП размером m выводящая шаблон.

На выход

Позиции вхождения шаблона в текст.

Алгоритмическая сложность

Алгоритм работает за $O(n^2 \cdot m)$ по времени и за $O(n \cdot m)$ по памяти.

Замечание 1

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

Замечение 2

Если ПП сбалансированны, то сложность работы алгоритма $O(n \cdot log(n) \cdot m)$.

- Реализация алгоритма Лифшица
- Оптимизация алгоритма и его параллельная версия
- Произведено сравнение с алгоритмом КМП

- Реализация алгоритма Лифшица
- Оптимизация алгоритма и его параллельная версия
- Произведено сравнение с алгоритмом КМП

- Реализация алгоритма Лифшица
- Оптимизация алгоритма и его параллельная версия
- Произведено сравнение с алгоритмом КМП

- Реализация алгоритма Лифшица
- Оптимизация алгоритма и его параллельная версия
- Произведено сравнение с алгоритмом КМП

Спасибо за внимание.