Задание 9

Приложение КС-грамматик для сжатия данных.

Преобразования КС-грамматик-1

Ключевые слова [1]:*язык, контекстно-свободный язык, магазинный автомат, грамматика, морфизм, метод математической индукции.*

Внимание: *Это двухнедельное задание. Оно может быть скорректированно, если мы что-то не успеем в следующую пятницу. Но все задачи из этого задания будут выданы.*

1 Приложение КС-грамматик для сжатия данных

Некоторые алгоритмы сжатия строк можно описать в терминах КС-грамматик. Мы рассмотрим два таких алгоритма. Первый из них носит название «Straight-line program» (SLP) и состоит в следующем. Слово w описывают с помощью КС-грамматики G_w, которая порождает единственное слово: $L(G_w) = w$. Грамматику G_w называют «Straight-line grammar» (SLG); этим же термином иногда называют и описываемый нами частный случай метода сжатия SLP: в роли программ выступают КС-грамматики.

Пример 1. Грамматика, описываемая правилами

$$S \to A_1A_1, \quad A_1 \to A_2A_2, \quad A_2 \to A_3A_3, \quad \dots \quad A_{n-1} \to A_nA_n, \quad A_n \to a,$$

порождает единственное слово a^{2^n}. Длина описания грамматики не превосходит cn, для некоторой константы $c > 0$, то есть имеет длину порядка логарифма от длины порождаемого слова, что является хорошим коэффициентом сжатия.

[1]минимальный необходимый объем понятий и навыков по этому разделу)

Задача 1. Постройте SLG G_n, порождающую слово

$$a^n b a^{n-1} b a^{n-2} b \dots ababa^2 ba^3 b \dots a^n b.$$

Длина описания G_n должна быть $cn, c > 0$. В качестве решения можно построить SLG G_5.

Замечание 1. *Преимуществом описанного метода сжатия является возможность эффективной проверки сжатого слова на регулярные события без разархивации. То есть существует алгоритм, получающий на вход описание НКА $\mathcal{A}$ и SLP G_w и проверяющий непустоту пересечения $L(\mathcal{A}) \cap L(G_w)$ за полиномиальное время от длин описаний $\mathcal{A}$ и G_w, но не w.*

Задача 2*. Постройте описанный выше алгоритм и докажите его корректность.

Мы описали общий метод сжатия SLP, но не описали пока алгоритма сжатия строк в грамматики. Таких алгоритмов существует несколько, одним из популярных алгоритмов сжатия такого типа является алгоритм Лемпеля–Зива–Велча (Lempel–Ziv–Welch, LZW). Опишем работу этого алгоритма на примере сжатия конкретной строки: $aababbbbaabaabab$.

Таблица 1:

Разбиение строки алгоритмом LZW

a	ab	abb	b	ba	aba	$abab$
A_1	A_2	A_3	A_4	A_5	A_6	A_7

Таблица 1 представляет собой словарь. Она устанавливает взаимно однозначное соответствие между нетерминалами и словами: слово w_i в построенной в итоге грамматике будет выводимо из A_i и только из A_i (но не обязательно за один шаг вывода). Опишем алгоритм заполнения таблицы-словаря.

1. В начале работы словарь пуст, слово u – необработанный суффикс слова w – совпадает с w, $i = 1$.

2. Алгоритм ищет максимальный префикс x необработанной части входа u, который был добавлен в словарь.

3. Если $u = xav, a \in \Sigma$, то алгоритм добавляет в словарь слово $w_i = xa$, удаляет префикс xa из u, увеличивает i на 1 и переходит к предыдущему шагу, если $u \neq \varepsilon$. Если же $u = \varepsilon$, алгоритм заканчивает работу.

4. Если $u = x$ и x уже соответствует некоторому нетерминалу A_j, то алгоритм добавляет в грамматику правило $A_i \to A_j$ и завершает работу. Обратим внимание, что x на этом шаге является суффиксом w.

Так, первая буква слова w всегда будет приписана нетерминалу A_1; далее в нашем примере за первой a идёт подслово ab, которое приписывается нетерминалу A_2, поскольку первая буква подслова a уже была приписана A_1; далее идёт подслово abb – подслово ab уже было приписано A_2 и т.д.

Нетрудно заметить, что искомая SLG имеет вид

$$S \to A_1 A_2 \dots A_7, \quad A_1 \to a, \quad A_2 \to A_1 b, \quad A_3 \to A_2 b,$$

$$A_4 \to b, \quad A_5 \to A_4 a, \quad A_6 \to A_2 a, \quad A_7 \to A_6 b.$$

Но как её эффективно построить алгоритмически, равно как и таблицу 1? Для этого воспользуемся техникой, базирующейся на конечных автоматах.

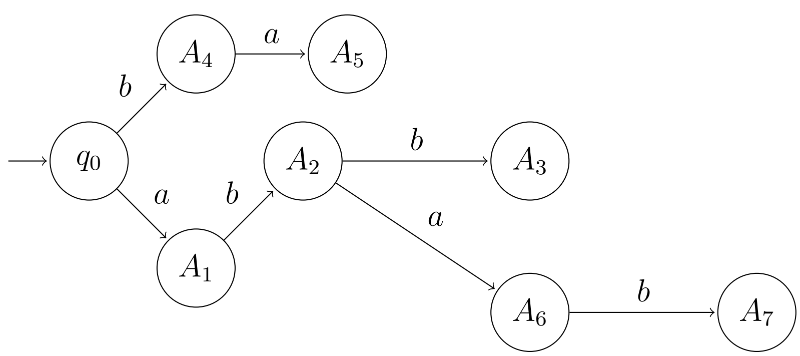


Рис. 2: LZW-автомат

В процессе построения SLG по алгоритму LZW мы строим LZW-автомат (рис. 2), который, по сути, реализует словарь. Однако, помимо стандартных функций словаря, LZW-автомат помечает каждую вершину, кроме начальной, нетерминалом A_i, устанавливая тем самым соответствие между словом w_i и состоянием автомата: $q_0 \xrightarrow{w_i} A_i$.

Алгоритм. В начале работы алгоритма словарь (реализуемый LZW-автоматом) пуст; обозначим через u_i необработанную часть входа – в начале работы $u_1 = w$.

Алгоритм находит кратчайший префикс $x_i \neq \varepsilon$ слова $u_i = x_i y_i$, которого ещё нет в словаре и добавляет его в словарь, помечая вершину, соответствующую этому слову новым нетерминалом A_i. Алгоритм повторяет этот процесс удалив из u_i префикс x_i ($u_{i+1} = y_i$) до тех пор пока либо слово u_{i+1} не окажется пустым, либо u_i не будет содержаться в словаре. При этом, если префикс x_i добавляется в словарь, то $x_i = va, a \in \Sigma$. В процессе добавления слов в словарь, алгоритм добавляет правила в грамматику согласно следующим правилам.

(1) Если $v = \varepsilon$, то алгоритм добавляет в грамматику правило $A_i \to a$.

(2) Если слово v непусто, то оно уже было добавлено в словарь и ему соответствует некоторый нетерминал A_j. Тогда алгоритм добавляет в грамматику правило $A_i \to A_j a$.

(3) Если слово u целиком содержится в словаре, то ему уже соответствует нетерминал A_j – в этом случае алгоритм добавляет правило $A_i \to A_j$.

(4) После окончания построения LZW-автомата, алгоритм добавляет к грамматике правило $S \to A_1 \ldots A_n$, где n – номер последнего добавленного нетерминала.

Корректность. Пусть $x_1, \ldots, x_{n-1}, (x_n)$ – последовательность слов добавленных в словарь в порядке добавления. Мы взяли x_n в скобки, поскольку последняя небработанная часть входа u_n могла целиком оказаться в словаре, тогда мы полагаем, что $x_n = u_n$. По построению, $w = x_1 x_2 \ldots x_n$, поскольку каждое слово x_i последовательно отрезали от w пока не осталось пустое слово.

Каждая вершина словаря, кроме q_0, помечена некоторым нетерминалом A_i. Докажем по индукции, что каждый нетерминал A_i порождает слово x_i, которое соответсвует вершине графа LZW-автомата.

База: $i = 1$. Поскольку словарь в начале не содержит слов, то первым добавленным словом x_1 будет первая буква слова w. Тогда алгоритм добавляет в грамматику правило $A_1 \to x_1$ согласно (1).

Шаг: от $i - 1$ к i. По предложению индукции, из каждого нетерминала A_j, $j < i$, выводится слово x_j. Для слова x_i возможно три случая. Первый: $x_i = a \neq x_j, \forall j < i$, здесь и далее $a \in \Sigma$; второй: $x_i = x_j a$ для некоторого $j < i$; третий: $x_i = x_j, j < i$ и необработанная часть слова пуста. В каждом из случаев утверждение для шага i верно согласно случаям добавлений правил (1)-(3) соответсвенно. Случай, когда слово x_i не было добавлено в словарь ранее и при этом $x_i \neq x_j a$, невозможен – алгоритм выбирает в качестве x_i кратчайший префикс необработанной части входа u_i, который не входит в словарь.

После того как мы доказали, что $A_i \Rightarrow^* x_i$, согласно (4) получаем, что $S \Rightarrow^* x_1 x_2 \ldots x_n = w$. □

Приведённый алгоритм очевидно работает за линейное время (от длины w). Строку, сжатую алгоритмом LZW легко декодировать как и строку, заданную произвольной SLG: нужно вывести единственную строку из грамматики, при этом каждый нетерминал раскрывается единственным образом. Также для алгоритма LZW справедливо замечание 1.

Задача 3. Постройте LZW-автомат и SLG G_w по описанному выше алгоритму для слова w:

а) $w = a^8$; 1. $w = tobeornottobeortobeornot$.

Задача 4. Постройте для слова $w = tobeornottobeortobeornot$ SLG, которая оптимальнее, чем построенная по алгоритму LZW. Численным показателем оптимальности является сумма длин правых частей всех правил SLG.

2 Приведённые КС-грамматики

Вообще говоря, не все нетерминалы из описания КС-грамматики могут встретиться в выводе некоторого слова. Такие нетерминалы могут возникнуть в ходе различных алгоритмических преобразований – ниже мы встретимся с такими преобразованиями. Для удобства, в частности для корректности работы многих алгоритмов, необходимо от таких бесполезных нетерминалов избавиться.

Выделяют два типа бесполезных нетерминалов. Нетерминал A называется *бесплодным*, если язык $L(G_A) = \{w \mid A \Rightarrow^* w\}$ пуст. Нетерминал A называется *недостижимым*, если ни одна цепочка вида $\alpha A \beta$ не выводится из S. Грамматика G называется *приведённой*, если она не содержит недостижимых и бесплодных нетерминалов.

Для того, чтобы удалить все бесплодные символы нужно действовать по следующему алгоритму:

- Множество $V_0 = T$.
- Множество V_{i+1} строим по V_i следующим образом. Положим в начале $V_{i+1} = V_i$. Если для правила $A \to \alpha$ справедливо $\alpha \in V_i^*$, то добавим нетерминал A в множество V_{i+1}.
- Как только $V_{i+1} = V_i$, объявляем $N = V_i \setminus T$, удаляем из P все правила, которые содержат нетерминалы не из V_i и заканчиваем работу.

Упражнение 1. Доказать корректность данного алгоритма.

Чтобы удалить все недостижимые символы нужно действовать по следующему алгоритму:

- Множество $V_0 = S$
- Множество V_{i+1} строим по V_i следующим образом. Положим в начале $V_{i+1} = V_i$. Если $A \in V_i$ и $A \to \alpha B \beta$, то добавим нетерминал B в множество V_{i+1}.
- Как только $V_{i+1} = V_i$, объявляем $N = V_i$, удаляем из P все правила, которые содержат нетерминалы не из V_i и заканчиваем работу.

Упражнение 2. Доказать корректность данного алгоритма.

Для того чтобы по грамматике G построить приведённую грамматику G', необходимо сначала удалить все бесплодные символы, а потом удалить все недостижимые символы. Действовать надо именно в таком порядке, потому что иначе после удаления бесплодных символов могут появится новые недостижимые символы, а после удаления недостижимых, новые бесплодные появиться не могут

3 Задачи

Задача 5(из к.д.з.). Решите уравнения с регулярными коэффициентами. В каждом пункте нужно выполнить три задания:

а) найти частное решение;

б) найти решение, минимальное по включению;

в) найти все решения.

1. $X = ((110)^* + 111^*)X.$

2. $X = (00 + 01 + 10 + 11)X + (0 + 1 + \varepsilon).$

3. $\begin{cases} Q_0 &= 0Q_0 + 1Q_1 + \varepsilon, \\ Q_1 &= 1Q_0 + 0Q_2, \\ Q_2 &= 0Q_1 + 1Q_2. \end{cases}$

Задача 6. $L = \{xcy \mid x, y \in \{a, b\}^*; x \neq y^R\}$. Постройте детерминированный МП-автомат, распознающий язык L. Если не получается построить детерминированный, постройте хотя бы недетерминированный.

Задача 7*. Пусть L – КС-язык. Докажите, что язык $\mathrm{Pref}(L) = \{u \mid \exists v \in \Sigma^* : uv \in L\}$, язык префиксов всех слов языка L, является КС-языком.

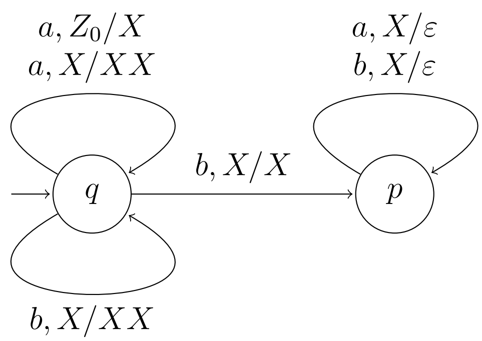
Задача 8. Постройте по грамматике G приведённую грамматику. Все построения должны быть выполнены строго по алгоритму. Грамматика G задана правилами:

$S \to A \mid B \mid C \mid E \mid AG$
$A \to C \mid aABC \mid \varepsilon$
$B \to bABa \mid aCbDaGb \mid \varepsilon$

$C \to BaAbC \mid aGD \mid \varepsilon$
$F \to aBaaCbA \mid aGE$
$E \to A$

Задача 9. МП-автомат M, допускающий по пустому стеку, задан диаграммой:

Постройте по МП-автомату M КС-грамматику G_M а по G_M приведённую КС-грамматику G.


$a, Z_0/X$
$a, X/XX$
$a, X/\varepsilon$
$b, X/\varepsilon$
$b, X/X$
q
p
$b, X/XX$