**Алгоритм Кока - Янгера - Касами**

Мы рассматриваем один из наиболее известных алгоритмов синтаксического анализа, применимый ко всему классу контекстно-свободных языков. Он может оказаться полезным в таких приложениях, когда исходные грамматики не обладают специальными свойствами, требуемых большинством специализированных алгоритмов. Например, если требуются неоднозначные грамматики и интерес представляют все разборы цепочки, как это бывает при работе с естественными языками.

**Постановка задачи**

**Определение.***Грамматикой* называется четверка $ G=(N,\Sigma,P,S)$, где

1. $ N$ -- конечное множество *нетерминальных символов*, или *нетерминалов* (иногда называемых вспомогательных символами, синтаксическими переменными или понятиями);
2. $ \Sigma$ -- не пересекающееся с $ N$ конечное множество *терминальных символов*, или *терминалов*;
3. $ P$ -- конечное подмножество множества

$\displaystyle {(N\cup\Sigma)}^{*} N {(N\cup\Sigma)}^{*} \times {(N\cup\Sigma)}^{*}
$

(элемент $ (\alpha,\beta)$ множества $ P$ называется *правилом* (или *продукцией* и записывается в виде $ \alpha\longrightarrow\beta$);

1. $ S$ -- выделенный символ из $ N$, называемый *начальным* (или *начальным*) *символом*.

**Определение.** *Грамматика $ G$* называется *контекстно-свободной* (или *бесконтекстной*), если каждое правило из $ P$ имеет вид $ A\longrightarrow\alpha$, где $ A\in N$, $ \alpha\in{(N\cup\Sigma)}^{*}$.

**Определение .**   Назовем контекстно-свободную грамматику   
$ G=(N,\Sigma,P,S)$ *грамматикой без $ e-$правил* (или *неукорачиваемой*), если либо

1. $ P$ не содержит $ e-$правил, либо
2. есть точно одно $ e-$правило $ S\longrightarrow e$ и $ S$ не встречается в правых частях остальных правил из $ P$.

**Определение .**   Контекстно-свободная грамматика $ G=(N,\Sigma,P,S)$ называется грамматикой в *нормальной форме Хомского* (или в *бинарной нормальной форме*, если каждое правило из $ P$ имеет один из следующих видов:

1. $ A\longrightarrow BC$, где $ A$,$ B и $$ C$ принадлежат $ N$,
2. $ A\longrightarrow a$, где $ a\in\Sigma$,
3. $ S\longrightarrow e$, если $ e\in L(G)$, причем $ S$ не встречается в правых частях правил.

В отличие от простейших алгоритмов нисходящего и восходящего разбора, алгоритм Кока - Янгера - Касами затрачивает на разбор время пропорциональное кубу длины входной цепочки и использует емность памяти пропорциональную квадрату длины входа.

Метод работает следующим образом. Пусть $ G=(N,\Sigma,P,S)$ -- контекстно-свободная грамматика без $ e-$правил в нормальной форме Хомского. Простое обобщение алгоритма работает и для грамматик не находящихся в этой нормальной форме.

Пусть $ \omega =a_1a_2...a_n$ -- входная цепочка, которую нужно разобрать согласно грамматике $ G$. Предполагается, что $ a_i\in\Sigma$ для $ 1\le i\le n$. Суть алгоритма состоит в построении треугольной *таблицы разбора T*, элементы которой обозначим $ t_{ij}$, где $ 1\le i\le n$ и$ 1\le j\le n-i+1$. Значениями переменных $ t_{ij}$ будут подмножества множества $ N$. Нетерминал $ A$ будет принадлежать $ t_{ij}$ тогда и только тогда, когда $ A{\Rightarrow}^{+}a_{i}a_{i+1}...a_{i+j-1}$, т.е. когда из $ A$ выводятся $ j$ входных символов, начиная с позиции $ i$. В частности, входная цепочка $ \omega$ принадлежит $ L(G)$ тогда и только тогда, когда $ S\in t_{1n}$.

Таким образом, чтобы выяснить, принадлежит ли $ \omega$ языку $ L(G)$, вычислим для ее $ \omega$ таблицу разбора $ T$ и посмотрим, принадлежит ли $ S$ ее элементу $ t_{1n}$. Затем, если нужен один или все разборы цепочки $ \omega$, их можно построить с помощью таблицы разбора. Для этой цели можно использовать в частности алгоритм *нахождения левого разбора по таблице разбора*[2](http://www.ctc.msiu.ru/program/t-system/diploma/footnode.html" \l "foot2702)[5].

**Алгоритм разбора Кока - Янгера - Касами**

*Bxoд*. Контекстно-свободная грамматика $ G=(N,\Sigma,P,S)$ в нормальной форме Хомского без $ e-$правил и входная цепочка $ \omega =a_1a_2...a_n\in{\Sigma}^{+}$.

*Выход*. Таблица разбора $ T$ для цепочки $ \omega$, такая, что $ A\in t_{ij}$ тогда и только тогда, когда $ A{\Rightarrow}^{+}a_{i}a_{i+1}...a_{i+j-1}$.

*Метод*.

1. Положить $ t_{i1}=\{A\vert A\longrightarrow a_i$ принадлежит $ P\}$ для каждого $ i$. После этого шага из $ A\in t_{i1}$ следует, очевидно, $ A{\Longrightarrow}^{+}a_i$.
2. Допустим, что уже вычислены $ t_{ij'}$ для всех $ 1\le i\le n$ и всех $ 1\le j' < j$. Положить $ t_{ij}=\{A$ для некоторого $ 1\le k < j$ правило $ A\longrightarrow BC$ принадлежит $ P,\: B\in t_{ik}$ и $ C\in t_{i+k, j-k}\}$. Так как $ 1\le k < j$, то $ k$ и $ j-k$ меньше $ j$. Таким образом, $ t_{ij}$ и$ t_{i+k,j-k}$ вычисляются раньше, чем $ t_{ij}$. После этого шага из $ A\in t_{ij}$ следует

$\displaystyle A\Longrightarrow BC{\Longrightarrow}^{+}a_{i}...a_{i+k-1}C{\Longrightarrow}^{+}a_{i}...a_{i+k-1}a_{i+k}...a_{i+j-1}.
$

1. Повторять шаг (2) до тех пор, пока не станут известны $ t_{ij}$ для всех $ 1\le i\le n$ и $ 1\le j\le n-i+1$.

В качестве примера рассмотрим грамматику $ G$ в нормальной форме Хомского с правилами

$\displaystyle S\longrightarrow AA \:\vert\:AS\:\vert\:b
$

$\displaystyle A\longrightarrow SA \:\vert\:AS\:\vert\:a
$

Пусть $ abaab$ - входная цепочка. Таблица разбор а $ T$, получающаяся в результате работы алгоритма, показана на рис. [20](http://www.ctc.msiu.ru/program/t-system/diploma/node39.html#parser).

|  |
| --- |
| \includegraphics[width=0.5\textwidth]{graphics/parser.eps} |
| **Рис. 20:** Таблица разбора T |

После шага (1) $ t_{11}=\{A\}$, так как $ A\longrightarrow a$ принадлежит $ P$ и $ a_1=a$. На шаге (2) в $ t_{32}$ добавляем $ S$, так как $ S\longrightarrow AA$ принадлежит $ P$ и $ A$ принадлежит $ t_{31}$ и $ t_{41}$. Заметим вообще, что, как видно из рисунка, $ t_{ij}$ для $ i>1$ можно вычислить обследовав нетерминалы в следующих парах элементов таблицы разбора:

$\displaystyle (t_{i1}, t_{i+1,j-1}), (t_{i2}, t_{i+2,j-2}), ... , (t_{i,j-1}, t_{i+j-1,1}).
$

Тогда, если $ B\in t_{ik}$ и $ C\in t_{i+k,j-k}$ для некоторого $ 1\le k < j$ и $ A\longrightarrow BC\in P$, добавляем $ A$ к $ t_{ij}$. Это значит, что мы одновременно движемся вверх по $ i-$му столбцу и вниз по диагонали, спускающейся вправо от ячейки $ t_{ij}$, обозревая нетерминалы, расположенные в проходимых таким образом парах ячеек.

Так как $ S\in t_{15}$, то $ abaab\in L(G)$.

.

**Модификация**

Алгоритм Кока-Янгера-Касами (Cocke — Younger — Kasami algorithm, CYK - алгоритм) - универсальный алгоритм, позволяющий по слову узнать, выводимо ли оно в заданной КС-грамматике в нормальной форме Хомского.

Пусть дана строка a_1 a_2 ... a_n. Заведем трехмерный массив d, состоящий из логических значений, и d[A,i,j] = true тогда и только тогда, когда из нетерминала A правилами грамматики можно вывести подстроку a_i a_{i+1} ... a_j. Тогда:

d[A,i,i] = true, если в грамматике присутствует правило A \rightarrow a_i, иначе false

Остальные элементы массива заполняются динамически: d[A,i,j] = \bigvee\limits_{A \rightarrow BC}\bigvee\limits_{k = i}^{j-1} d[B,i,k] \wedge d[C,k+1,j]. То есть, подстроку a_i...a_j можно вывести из нетерминала A, если существует продукция A \rightarrow BC и такое k, что подстрока a_i...a_k выводима из B, а подстрока a_{k+1}...a_j - из C.

Значение d[S,1,n] содержит ответ на вопрос, выводима ли данная строка в данной грамматике.

Заметим, что если массив будет хранить целые числа, а формулу заменить на d[A,i,j] = \sum\limits_{A \rightarrow BC}\sum\limits_{k = i}^{j-1} d[B,i,k] \cdot d[C,k+1,j], то d[A,i,j] - количество способов получить подстроку a_i...a_j из нетерминала A.

Пусть P_{A \rightarrow BC} - стоимость вывода по правилу A \rightarrow BC. Тогда, если использовать формулу d[A,i,j] = \min\limits_{A \rightarrow BC} \min\limits_{k = i}^{j-1}  ( d[B,i,k] + d[C,k+1,j] + P_{A \rightarrow BC} ), то d[A,i,j] - минимальная стоимость вывода подстроки a_i...a_j из нетерминала A.

Таким образом, задача о выводе в КС-грамматике в нормальной форме Хомского является обобщением задачи динамического программирования на подотрезке.

**Сложность алгоритма**

Пусть, n - длина входной строки, а m - количество правил вывода в грамматике.

Обработка правил вида A \rightarrow a_i выполняется за O(nm).

Проход по всем подстрокам выполняется за O(n^2). В обработке подстроки присутствует цикл по всем правилам вывода и по всем разбиениям на две подстроки, следовательно обработка работает за O(nm). В итоге - O(n^3 m).

Следовательно, общее время работы алгоритма - O(n^3 m). Кроме того, алгоритму требуется память (на массив d) объемом O(n^2 m).

**Псевдокод**

function CYK (a - строка длины n, G - набор правил вывода грамматики с m нетерминалами, S - стартовый нетерминал) -> bool

begin

d : array [1..m,1..n,1..n] of bool

for i = 1 to n

if (A -> a[i] - продукция)

d[A,i,i] = true

for len = 1 to n-1

for i = 1 to n-l

for (A -> BC - продукция)

for k = i to i+len-1

d[A,i,i+len] = d[A,i,i+len] or (d[B,i,k] and d[C,k+1,i+len])

return d[S,1,n]

end