В работе (Охотин) предложен алгоритм ....

АЛГОРТМ

Далее предложена модификация данного алгоритма.

Введем обозначения, немного отличающиеся от используемых в статье (Охотин, 2013). Пусть $G = (\Sigma, N, R, S)$ — грамматика и $w = a_0 \dots a_{n-1}$ — строка, причем $n+1 = 2^k$. T — треугольная матрица $n \times n$ с элементами $T[i,j] \subseteq N$. Цель алгоритма — заполнить ячейки матрицы T так, чтобы выполнялось следующее условие:

$$T[i,j] = \{A \mid a_{n-1-j} \dots a_i \in L(A)\}, \quad \text{при } 0 \leqslant i,j \leqslant n-1 \leqslant i+j\,.$$

Вторая матрица P с элементами $P[i,j] \subseteq N \times N$ в результате должна быть заполнена значениями

$$P[i,j] = \{(B,C) \mid a_{n-1-j} \dots a_i \in L(B)L(C)\}, \quad 0 \leqslant i,j \leqslant n-1 \leqslant i+j\,.$$

Определение 0.1. Назовем *(квадратной) подматрицей* такой набор пар индексов $S = \{(i,j)\}$, что для каких-то $0 \leqslant a,b \leqslant n-1 \leqslant a+b$ и $0 < size \leqslant n-a, n-b$ любая пара (i,j), удовлетворяющая условиям $a \leqslant i < a+size$ и $b \leqslant j < b+size$, принадлежит множеству S. Тогда $size$ — *размер*, а пара индексов (a,b) — *вершина* этой подматрицы.

Определение 0.2. Пусть S — подматрица. Будем обозначать через $T[S]$ и $P[S]$ подматрицы матриц T и P, соответствующие множеству индексов S.

Для описания алгоритма нам в первую очередь необходимо задать структуру для представления подматриц и определить для этой структуры следующие функции.

Listing 1: Submatrix helpers.

```
1 function size(submatrix)
2 function vertex(submatrix)
3
4 function leftSubmatrix(submatrix)
5 function topSubmatrix(submatrix)
6 function rightSubmatrix(submatrix)
7 function bottomSubmatrix(submatrix)
```

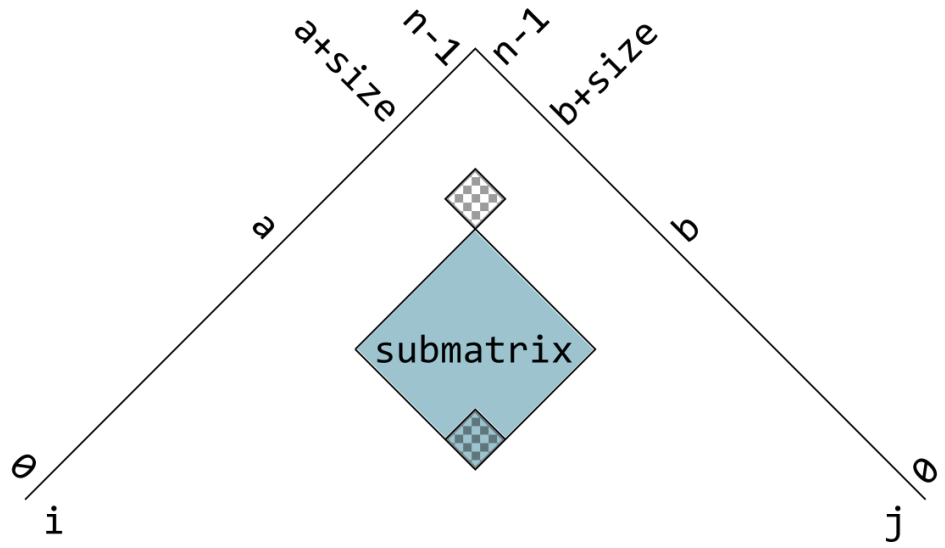
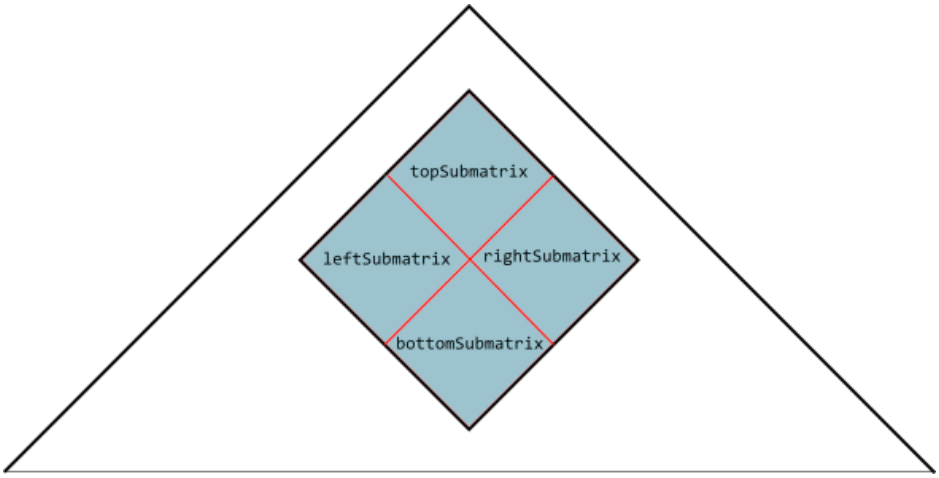


Рис. 1: Иллюстрация к определению 0.1

```
8
9  function rightNeighbor(submatrix)
10 function leftNeighbor(submatrix)
11 function rightGrounded(submatrix)
12 function leftGrounded(submatrix)
```

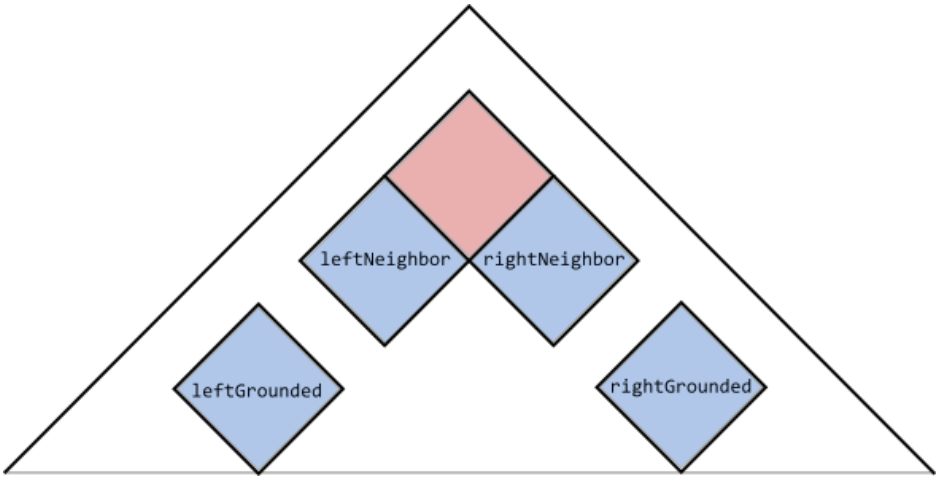
Здесь, функции size и vertex вычисляют соответственно размер и вершину подматрицы. Следующие четыре функции leftSubmatrix — bottomSubmatrix возвращают одну из подматриц, делящих исходную подматрицу на четыре части, как показано на рисунке 2.

Рис. 2: Иллюстрация к листингу



В свою очередь функции rightNeighbor — leftGrounded возвращают подматрицы, сдвинутые относительно исходной так, как показано на рисунке 3.

Рис. 3: Иллюстрация к листингу

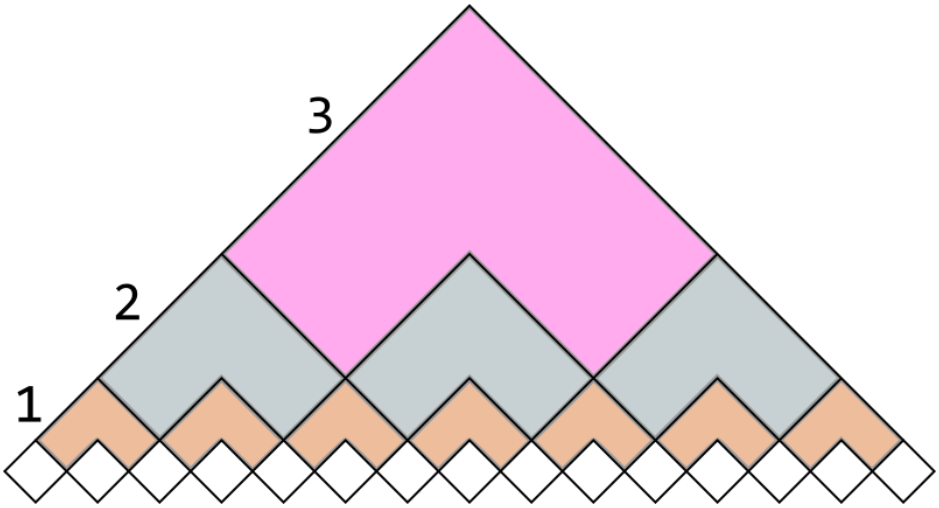


Теперь можно приступить к описанию самого алгоритма. Он состоит из трех процедур.

Главная процедура main сначала обрабатывает нижний слой матрицы T (клетки (i, j), для которых $i + j = n - 1$), записывая в него коррект-

ные значения. Потом разбивает матрицу T на слои, как показано на картинке 4 (каждый слой состоит из набора подматриц, у каждой из которых отброшена нижняя четверть — bottomSubmatrix). Полученные слои обрабатываются последовательно снизу вверх, с помощью процедуры completeVLayer, заполняя тем самым всю матрицу T.

Рис. 4: Первичное разбиение на слои



Процедура completeVLayer, в свою очередь, принимает на вход набор подматриц M. Для каждого элемента m этого набора процедура достраивает матрицу T для трех верхних четвертей (leftSubmatrix(m), rightSubmatrix(m) и topSubmatrix(m)). Для корректной работы этой функции необходимо, чтобы для любой $m \in M$ ячейки $T[i, j]$ были построены для $(i, j) \in$ bottomSubmatrix(m), а также для таких (i, j), что $i < a, j < b + s$ или $i < a + s, j < b$, где (a, b) — вершина, а s — размер подматрицы m.

Третья процедура — completeLayer — тоже принимает на вход набор подматриц M, но для каждого элемента m этого набора достраивает матрицу T для всей подматрицы m. Для корректной работы этой функции требуется, чтобы для любой $m \in M$ ячейки $T[i, j]$ были построены для таких (i, j), что $i < a, j < b + s$ или $i < a + s, j < b$, где (a, b) — вершина, а s — размер подматрицы m.

Listing 2: Algorithm.

```
1  procedure main():
2    for ℓ in (0, ..., n−1) do
3      T[ℓ, n−1−ℓ] = {A | A → a_ℓ ∈ R}
4    for i in 1 .. k−1 do
5      denote layer = constructLayer(i)
6      completeVLayer(layer)
7
8  procedure completeLayer(M):
9    if m ∈ M and size(m) = 1 then
10     denote cells = {vertex(m) | m ∈ M}
11     foreach (x, y) ∈ cells do
12       T[x, y] = f(P[x, y])
13   else
14     denote bottomLayer = {bottomSubmatrix(m) | m ∈ M}
15     completeLayer(bottomLayer)
16     completeVLayer(M)
17
18 procedure completeVLayer(M):
19   denote leftSubLayer = {leftSubmatrix(m) | m ∈ M}
20   denote rightSubLayer = {rightSubmatrix(m) | m ∈ M}
21   denote topSubLayer = {topSubmatrix(m) | m ∈ M}
22
23   foreach m ∈ leftSubLayer do
24     P[m] = P[m] ∪ (T[leftGrounded(m)] × T[rightNeighbor(m)])
25   foreach m ∈ rightSubLayer do
26     P[m] = P[m] ∪ (T[leftNeighbor(m)] × T[rightGrounded(m)])
27
28   completeLayer(leftSubLayer ∪ rightSubLayer)
29
30   foreach m ∈ topSubLayer do
31     P[m] = P[m] ∪ (T[leftGrounded(m)] × T[rightNeighbor(m)])
32   foreach m ∈ topSubLayer do
33     P[m] = P[m] ∪ (T[leftNeighbor(m)] × T[rightGrounded(m)])
34
35   completeLayer(topSubLayer)
```

Процедура main реализуется через completeVLayer очевидным образом. Процедура completeLayer по сути аналогична процедуре complete из статьи, за исключением того, что она выполняется сразу для нескольких матриц. Таким же образом отдельно разбирается случай, когда пере-

данные матрицы имеют размер 1×1. Иначе матрицы разбиваются на четыре части и сначала следует рекурсивный вызов от bottomSubmatrix (для всех переданных матриц), а затем вызов процедуры completeVLayer, которая обрабатывает верхние части матриц.

Процедура completeVLayer для каждой из переданных матриц сначала выполняет два перемножения (соответствует 14 и 16 строкам в алгоритме из статьи), затем вызывает completeLayer от rightSubmatrix и leftSubmatrix (15 и 17 строки оригинального алгоритма), далее выполняет оставшиеся два умножения (строки 18 и 19) и, наконец, вызывает completeLayer от оставшейся части topSubmatrix (строка 20).

Представленная модификация позволяет ....