

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, что будет выполняться

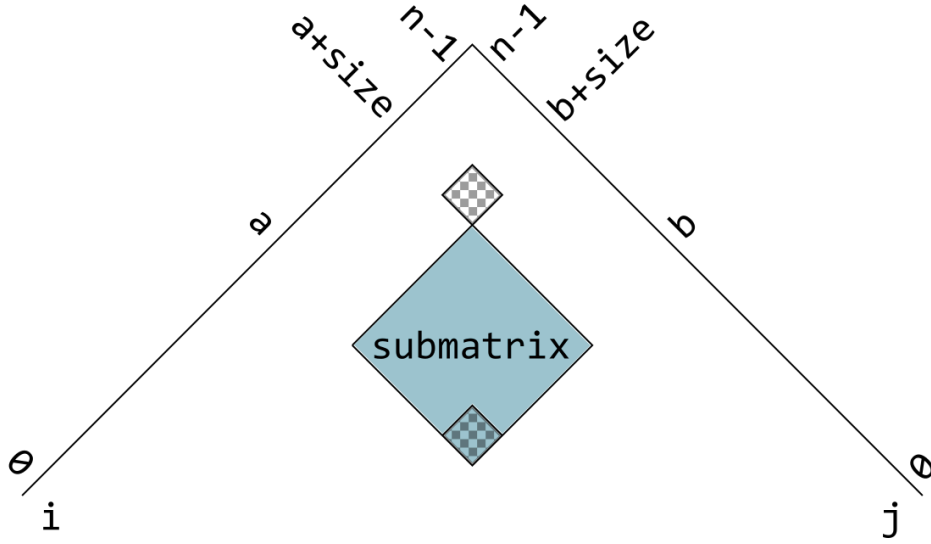
$$T[i, j] = \{A \mid a_{n-1-j} \dots a_i \in L(A)\}, \quad \text{при } 0 \leq i, j \leq n-1 \leq i+j.$$

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

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

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

Рис. 1: Иллюстрация к определению.



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

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

Листинг 1: Submatrix helpers.

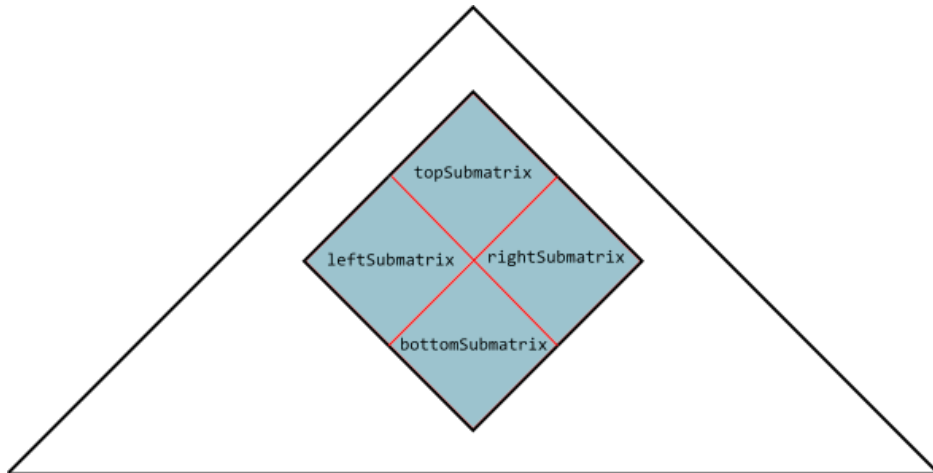
---

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

---

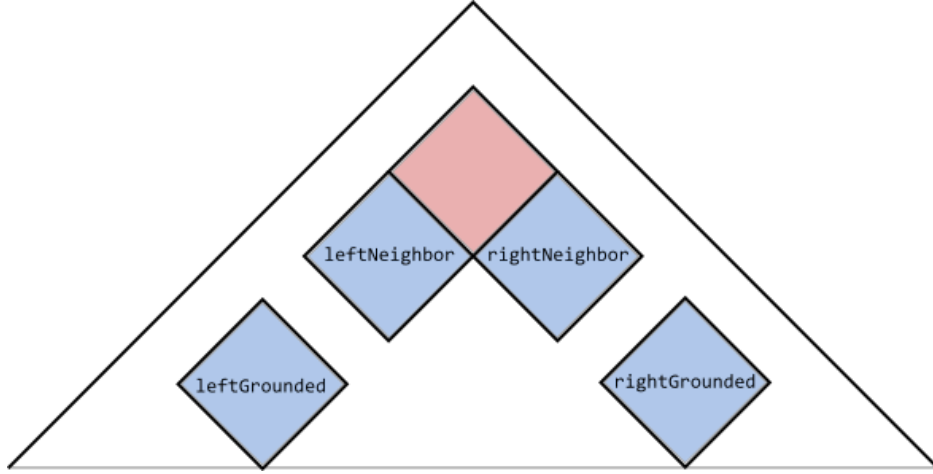
Здесь, функции `size` и `vertex` возвращают соответственно размер и вершину подматрицы. Следующие четыре функции `leftSubmatrix` – `bottomSubmatrix` возвращают одну из подматриц, делящих исходную подматрицу на четыре части, как показано на рисунке 2.

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.



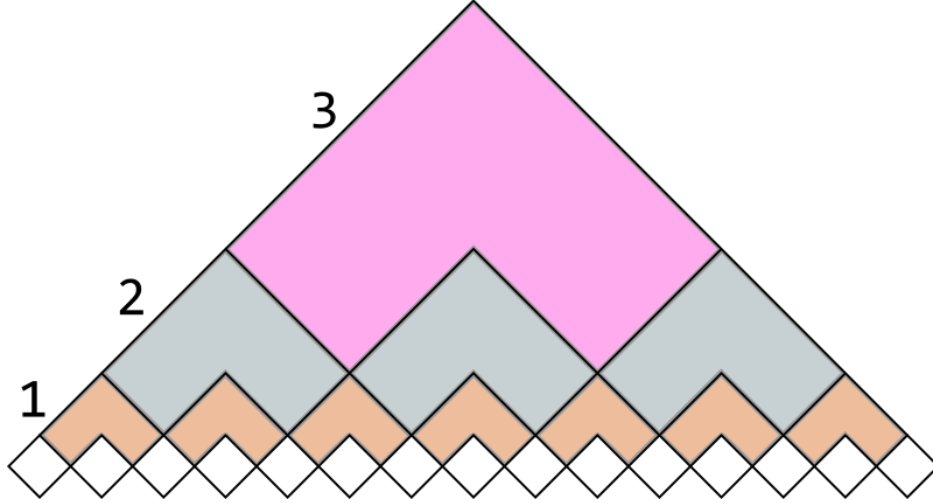
Теперь можно приступить к описанию самого алгоритма. Он состоит из трех процедур.

Главная процедура `main` сначала обрабатывает нижний слой матрицы  $T$  (клетки  $(i, j)$ , для которых  $i + j = n - 1$ ), записывая в него корректные значения. Потом разбивает матрицу  $T$  на слои, как показано на картинке 4 (каждый слой по сути 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  $T$ .

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  $(i, j) \in \text{bottomSubmatrix}(m)$ , 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 под-

Рис. 4: Первичное разбиение на слои.



матриц  $M$ , 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  $m$ .

Листинг 2: Algorithm.

---

```

1 procedure main():
2   for  $\ell$  in  $(0, \dots, n-1)$  do
3      $T[\ell, n-1-\ell] = \{A \mid A \rightarrow a_\ell \in R\}$ 
4   for  $i$  in  $1 \dots k-1$  do
5     denote layer = constructLayer( $i$ )
6     completeVLayer(layer)
7
8 procedure completeLayer( $M$ ):
9   if  $m \in M$  and  $\text{size}(m) = 1$  then
10    denote cells =  $\{\text{vertex}(m) \mid m \in M\}$ 
11    foreach  $(x, y) \in \text{cells}$  do
12       $T[x, y] = f(P[x, y])$ 
13  else
14    denote bottomLayer =  $\{\text{bottomSubmatrix}(m) \mid m \in M\}$ 
15    completeLayer(bottomLayer)
16    completeVLayer( $M$ )

```

```

17
18 procedure completeVLayer( $M$ ):
19   denote leftSubLayer = {leftSubmatrix( $m$ ) |  $m \in M$ }
20   denote rightSubLayer = {rightSubmatrix( $m$ ) |  $m \in M$ }
21   denote topSubLayer = {topSubmatrix( $m$ ) |  $m \in M$ }
22
23   foreach  $m \in$  leftSubLayer do
24      $P[m] = P[m] \cup (T[\text{leftGrounded}(m)] \times T[\text{rightNeighbor}(m)])$ 
25   foreach  $m \in$  rightSubLayer do
26      $P[m] = P[m] \cup (T[\text{leftNeighbor}(m)] \times T[\text{rightGrounded}(m)])$ 
27
28   completeLayer(leftSubLayer  $\cup$  rightSubLayer)
29
30   foreach  $m \in$  topSubLayer do
31      $P[m] = P[m] \cup (T[\text{leftGrounded}(m)] \times T[\text{rightNeighbor}(m)])$ 
32   foreach  $m \in$  topSubLayer do
33      $P[m] = P[m] \cup (T[\text{leftNeighbor}(m)] \times T[\text{rightGrounded}(m)])$ 
34
35   completeLayer(topSubLayer)

```

---

Процедура main реализуется через completeVLayer очевидным образом. 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 переданные матрицы имеют размер один. Иначе матрицы разбиваются на четыре части и сначала следует рекурсивный вызов от bottomSubmatrix (для всех переданных матриц), а потом вызов процедуры completeVLayer, которая обрабатывает верхние части матриц.

completeVLayer для каждой из переданных матриц сначала выполняет два перемножения (соответствует 14 и 16 строкам в алгоритме из статьи), потом 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).