# Модификация алгоритма Валианта для задачи поиска подстрок

[1] *Ю.А. Сусанина <st049970@student.spbu.ru>*
[2] *А.Н. Явейн <yaveyn@yandex.ru>*
[3] *С.В. Григорьев <s.v.grigoriev@spbu.ru>*
[1, 3] *Санкт-Петербургский государственный университет,*
*199034, Россия, г. Санкт-Петербург, Университетская наб., д. 7/9.*

**Аннотация**.

Теория формальных языков и, в частности, контекстно-свободные (КС) грамматики активно изучаются и находят широкое применение во многих областях. Например, в биоинформатике в задачах распознавания и классификации иногда необходимо найти подпоследовательности генетических цепочек, обладающие некоторыми характерными чертами, которые могут быть описаны с помощью грамматики. Задача поиска этих подпоследовательностей сводится к проверке их принадлежности языку, заданному грамматикой. Такие области, как биоинформатика, требуют работы с большими объёмами данных, что приводит к необходимости усовершенствования существующих методов синтаксического анализа. На данный момент среди алгоритмов синтаксического анализа, работающих с любой КС-грамматикой, самым асимптотически эффективным является алгоритм Валианта, основанный на использовании матричных операций. В данной работе предложена модификация данного алгоритма, основным достоинством которой является возможность разбиения матрицы разбора на подслои непересекающихся подматриц, которые могут быть обработаны независимо. Предложенная версия алгоритма легко адаптируется к задаче поиска подстрок. Проведенные эксперименты показывают, что модификация сохранила основные преимущества исходного алгоритма, главное из которых – высокая производительность, полученная за счет использования эффективных методов перемножения матриц. Также модифицированная версия позволила заметно уменьшить время, затрачиваемое на поиск подстрок, сократив большое количество избыточных вычислений.

**Ключевые слова:** синтаксический анализ; контекстно-свободные грамматики; матричные операции.

**DOI: ???**

**Для цитирования:** Сусанина Ю.А., Явейн А.Н., Григорьев С.В. Заголовок статьи. Труды ИСП РАН, том ?, вып. ?, 2019 г., стр. ?-?. DOI: ???

## 1. Введение

После того, как Ноам Хомский внес существенный вклад в развитие теории формальных языков, выделенный им тип грамматик — контекстно-свободные (КС) — начали активно изучаться и в дальнейшем нашли широкое применение во многих областях [2], прежде всего, в информатике — для описания естественных языков и языков программирования. Также существует множество исследований, которые показывают эффективность использования КС-грамматик в биоинформатике [12, 13].

Хорошим примером является возможность применения формальных языков для решения задач распознавания и классификации, некоторые из которых основаны на том, что вторичная структура последовательностей ДНК и РНК содержит в себе важную информацию об организме. Характерные особенности вторичной структуры могут быть описаны с помощью КС-грамматики. Это позволяет свести проблемы распознавания и классификации к задаче синтаксического анализа (определения принадлежности некоторой строки к языку, заданному грамматикой). То есть наличие подпоследовательностей, обладающих некоторыми особенностями, а также их расположение относительно друг друга помогают получить информацию о происхождении организма. Основной задачей было найти алгоритм синтаксического анализа, легко адаптируемый к задаче поиска подстрок, и, кроме того, найденное решение должно быть максимально эффективным, так как такая область применения, как биоинформатика, предполагает работу большими объемами данных.

Большинство алгоритмов синтаксического анализа либо работают за кубическое время (Касами [3], Янгер [4], Эрли [10]), либо применяются только к определенным подклассам КС-грамматик (Бернарди, Клауссен [11]). На данный момент самым асимптотически эффективным алгоритмом синтаксического анализа, работающим с любой КС-грамматикой, является алгоритм Валианта [5]. Результатом его работы для линейного входа является матрица разбора, каждый элемент которой отвечает за выводимость конкретной подстроки. Валиант смог добиться улучшения вычислительной сложности за счет использования матричных операций. Для входной строки длины n алгоритм заканчивает свою работу за время $O(BMM(n)log(n))$, где $BMM(n)$ — время, необходимое для перемножения двух булевых матриц размера $n \times n$. Более того, данный данный алгоритм был расширен для конъюнктивных и булевых грамматик, которые обладают большей выразительностью [1,7,8]. Однако алгоритм Валианта плохо применим к описанной выше проблеме поиска подстрок, так как он будет выполнять много лишних вызовов перемножения матриц.

В данной работе предложена модификация алгоритма Валианта. За счет изменения порядка вычисления перемножений матриц появилась возможность разбиения матрицы разбора на слои непересекающихся подматриц. Предложенный подход частично решает проблему поиска подстрок. Кроме

того, каждая матрица слоя может обрабатываться независимо, что в дальнейшем позволит повысить эффективность алгоритма, используя техники параллельных вычислений.

Работа организована следующим образом. В разделе 2 даны основные понятия и приведен исходный алгоритм Валианта; в разделе 3 представлена модификация данного алгоритма, легко адаптируемая к задаче поиска-подстрок и позволяющая повысить использование параллельных техник, а также доказана корректность и приведена оценка сложности модифицированный версии; в разделе 4 показана применимость предложенного нами подхода к задаче поиска подстрок; в разделе 5 представлены результаты проведенных экспериментов; заключение и направления будущих исследований приведены в разделе 6.

## *2. Обзор*

В этом разделе мы введем основные определения и алгоритм Валианта, на котором основывается предложенная в данной работе модификация.

### 2.1 Терминология

Грамматикой будем называть четверку $(\Sigma, N, R, S)$, $\Sigma$ – конечное множество терминальных символов, $N$ – конечное множество нетерминальных символов, $R$ –конечное множество правил вида $\alpha \to \gamma$, где $\alpha \in V^*NV^*, \gamma \in V^*, V = \Sigma \cup N$ и $S \in N$ – стартовый символ.

Грамматика называется контекстно-свободной (КС), если любое ее правило $r \in R$ имеет вид $A \to \beta$, где $A \in N, \beta \in V^*$.

КС-грамматика $G = (\Sigma, N, R, S)$ называется грамматикой в нормальной форме Хомского, если любое ее правило имеет одну из следующих форм:

- $A \to BC$,
- $A \to a$,
- $S \to \varepsilon$ (если пустая строка $\varepsilon \in L_G$),

где $A, B, C \in N$, $a \in \Sigma$, $L_G$ – язык, порождаемый грамматикой $G$.

Как $L_G(A)$ будем обозначать язык, порождаемый грамматикой $G_A = (\Sigma, N, R, A)$.

### 2.2 Алгоритм синтаксического анализа, основанный на перемножении матриц

Задачей синтаксического анализа является проверка принадлежности входной строки языку, порождаемому некоторой грамматикой.

Взятый за основу в данной статье алгоритм Валианта относится к табличным методам синтаксического анализа, главная идея которых – построение для входной строки $a = a_1 \dots a_n$ и КС-грамматики в нормальной форме Хомского

$G = (\Sigma, N, R, S)$ таблицы (далее матрицы) разбора $T$ размера $(n+1) \times (n+1)$, где

$$T_{i,j} = \{A: A \in N, a_{i+1} \dots a_j \in L_G(A)\} \text{ для всех } i < j.$$

Элементы матрицы $T$ должны заполняться последовательно, начиная с диагонали:

$$T_{i-1,i} = \{A: A \to a_i \in R\}.$$

Затем $T_{i,j}$ будут вычисляться по формуле $T_{i,j} = f(P_{i,j})$, где

$$P_{i,j} = \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$$

$$f(P_{i,j}) = \{A | \exists A \to BC \in R, (B,C) \in P_{i,j}\}.$$

Входная строка $a = a_1 \dots a_n$ принадлежит языку $L_G$ тогда и только тогда, когда $S \in T_{0,n}$.

Сначала введем понятие перемножения двух подматриц матрицы разбора $T$.

Если все элементы данной матрицы заполнять последовательно, то вычислительная сложность данного алгоритма будет равна $O(n^3)$. Наиболее затратной по времени операцией является вычисление $\bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$, и Валиант смог реорганизовать порядок заполнения элементов матрицы разбора так, что стало возможным перенести эти вычисления на перемножение булевых матриц.

Пусть $X \in (2^N)^{m \times l}, Y \in (2^N)^{l \times n}$ – две подматрицы $T$, тогда $X \times Y = Z$, где $Z \in (2^N)^{m \times n}$ и $Z_{i,j} = \bigcup_{k=1}^{l} X_{i,k} \times Y_{k,j}$.

Теперь можно представить $X \times Y = Z$ как перемножение $|N|^2$ булевых матриц (для каждой пары нетерминалов). Определим матрицу, соответствующую паре нетерминалов $(B, C)$, как $Z^{(B,C)}$. Тогда $Z_{i,j}^{(B,C)} = 1$ тогда и только тогда, когда $(B, C) \in Z_{i,j}$. Также заметим, что $Z^{(B,C)} = X^B \times Y^C$. Более того, каждое из перемножений булевых матриц может быть обработано независимо.

Эти изменения позволили добиться сложности $O(BMM(n)log(n))$, где $BMM(n)$ — время, необходимое для перемножения двух булевых матриц размера $n \times n$.

Алгоритм Валианта представлен на листинге 1. Все элементы матриц $T$ и $P$ инициализируются пустыми множествами. Затем эти элементы последовательно заполняются двумя рекурсивными процедурами.

Процедура $compute(l, m)$ корректно заполняет все $T_{i,j}$ для всех $l \leq i < j < m$.

Процедура $complete(l, m, l', m')$ заполняет все $T_{i,j}$ для всех $l \leq i < m$ и $l' \leq j < m'$. Для корректной работы этой процедуры, во-первых, необходимо, чтобы элементы $T_{i,j}$ для всех $i$ и $j$, таких что $l \leq i < j \leq m$ и $l' \leq i < j \leq m'$

уже были построены. Во-вторых, текущее значение $P_{i,j}$ для всех $i$ и $j$, таких что $l \le i < m$ и $l' \le j < m'$, должно быть следующим:

$$P_{i,j} = \{(B,C): \exists k, (m \le k < l'), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}.$$

Входные данные: G – КС-грамматика, $a = a_1 \dots a_n$, $a_i \in \Sigma, n \ge 1$, где $n+1$ – степень двойки

main():
$compute(0, n+1)$;
accept if and only if $S \in T_{0,n}$
compute$(l, m)$:
**if** $m - l \ge 4$ **then**
$compute(l, \frac{l+m}{2})$;
$compute(\frac{l+m}{2}, m)$
$complete\ (l, \frac{l+m}{2}, \frac{l+m}{2}, m)$
complete$(l, m, l', m')$:
**if** $m - l = 1$ **and** $m = l'$ **then**
$T_{i-1,i} = \{A: A \to a_i \in R\}$;
**else if** $m - l = 1$ **and** $m < l'$ **then**
$T_{i,j} = f(P_{i,j})$;
**else if** $m - l > 1$ **then**
$leftgrounded = \left(l, \frac{l+m}{2}, \frac{l+m}{2}, m\right)$;$rightgrounded = \left(l', \frac{l\prime+m\prime}{2}, \frac{l\prime+m\prime}{2}, m'\right)$;
$bottom = (\frac{l+m}{2}, m, l', \frac{l\prime+m\prime}{2})$; $left = (l, \frac{l+m}{2}, l', \frac{l\prime+m\prime}{2})$;
$right = (\frac{l+m}{2}, m, \frac{l\prime+m\prime}{2}, m')$; $top = (l, \frac{l+m}{2}, \frac{l\prime+m\prime}{2}, m')$;
$complete(bottom)$;
$P_{left} = P_{left} \cup (T_{leftgrounded} \times T_{bottom})$;
$complete(left)$;
$P_{right} = P_{right} \cup (T_{bottom} \times T_{rightgrounded})$;
$complete(right)$;
$P_{top} = P_{top} \cup (T_{leftgrounded} \times T_{right})$;
$P_{top} = P_{top} \cup (T_{left} \times T_{rightgrounded})$;
$complete(top)$

*Листинг 1. Алгоритм Валианта*

*Listing 1. Valiant's algorithm*

Деление на подматрицы во время выполнения этой процедуры показано на рис. 1.

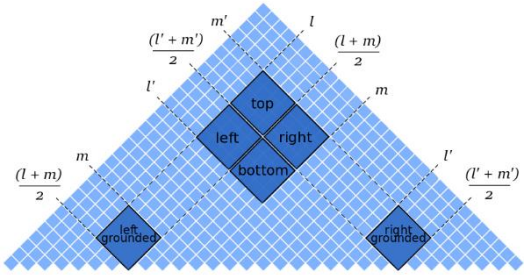


*Рис. 1. Деление матриц, использованное в процедуре* $complete(l, m, l', m')$

*Fig. 1. Matrix partition used in* $complete(l, m, l', m')$ *procedure*

На рис. 2 представлен фрагмент процесса заполнения матрицы разбора. В следующем разделе будет приведен пример работы модификации, который наглядно продемонстрирует разницу и преимущества предложенной нами версии.

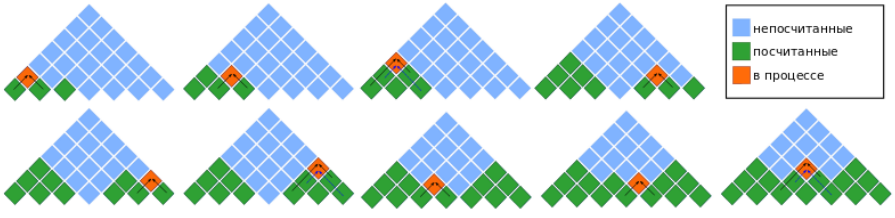


*Рис. 2. Пример работы алгоритма Валианта*

*Fig. 2. An example of Valiant's algorithm*

## 3. Модификация

В данном разделе мы представляем модификацию алгоритма Валианта. За счет изменения порядка вычисления подматриц в исходном алгоритме модифицированная версия обладает такими практическими преимуществами, как адаптация для решения задачи поиска подстрок и упрощение использования параллельных вычислений.

Главное отличие предложенной модификации – это возможность разделения матрицы разбора на слои непересекающихся подматриц одинакового размера. Пример разбиения матриц на такие слои представлен на рис. 3. Каждый слой состоит из квадратных подматриц, размер которых равен степени двойки. Слои заполняются последовательно снизу вверх, и каждая матрица слоя может

обрабатываться независимо, что позволяет значительно упростить разработку параллельной версии алгоритма.

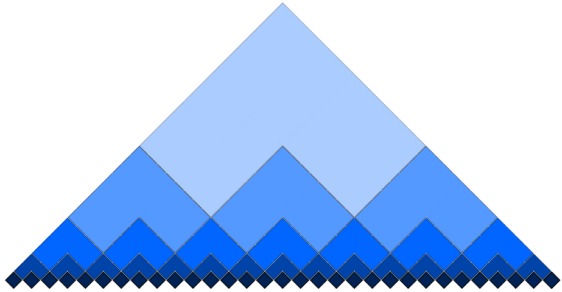

*Рис. 3. Деление матриц на V-образные слои*

*Fig. 3. Matrix partition on V-shaped layers*

Пример работы модификации показан на рис. 4. Нижний слой (из подматриц размера 1) вычисляется заранее, и заполнение матрицы начинается со второго слоя. (Здесь и далее, под слоем будем понимать некоторое множество подматриц матрицы разбора.)

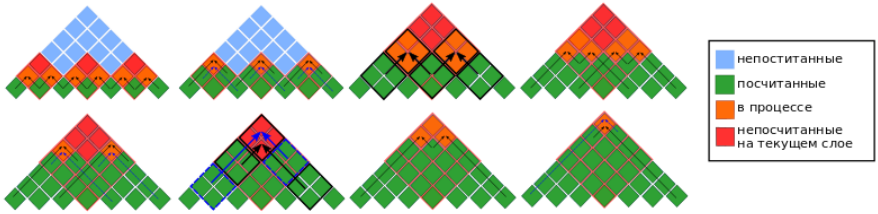


*Рис. 4. Пример работы модификации алгоритма Валианта*

*Fig. 4. An example of the modification of Valiant's algorithm*

Модифицированная версия алгоритма представлена на листинге 1. Процедура $main()$ заполняет нижний слой матрицы ($T_{l,l+1}$), а затем разделяет матрицу на слои, так, как было описано ранее, которые корректно вычисляются в процедуре $completeVLayer()$. Таким образом, вызов функции $main()$ заполнит всю матрицу разбора и вернет информацию о выводимости строки для заданной грамматики.

Для краткости, определим несколько дополнительных функций: $left(subm)$, $right(subm)$, $bottom(subm)$, $top(subm)$, $rightgrounded(subm)$ and $leftgrounded(subm)$, которые возвращают подматрицы для матрицы $subm =$

$(l, m, l', m')$, аналогично разбиению матрицы в алгоритме Валианта, который был представлен на рис. 2.

Также определим функции для слоя подматриц $M$:

- $bottomsublayer(M) = \{bottom(subm) \mid subm \in M\}$,
- $leftsublayer(M) = \{left(subm) \mid subm \in M\}$,
- $rightsublayer(M) = \{right(subm) \mid subm \in M\}$,
- $topsublayer(M) = \{top(subm) \mid subm \in M\}$.

Входные данные: G – КС-грамматика, $a = a_1 \dots a_n$, $a_i \in \Sigma, n \geq 1$, где $n + 1 = 2^p$

main():
**for** $l \in \{1, \dots, n\}$ **do**
  $T_{l,l+1} = \{A: A \to a_i \in R\}$
**for** $l \leq i < p - 1$ **do**
  $completeVLayer(constructLayer(i))$
accept if and only if $S \in T_{0,n}$

constructLayer($i$):
$\{(k2^i, (k+1)2^i, (k+1)2^i, (k+2)2^i) \mid 0 \leq k < 2^{p-i} - 1\}$

completeLayer($M$):
**if** $\forall (l, m, l', m') \in M\ (m - l = 1)$ **then**
  **for** $(l, m, l', m') \in M$ **do**
    $T_{l,l'} = f(P_{l,l'})$
**else**
  $completeLayer(bottomsublayer(M))$;
  $completeVLayer(M)$

complete$VLayer(M)$:
$multiplicationTask_1 =$
  $\{left(subm), leftgrounded(subm), bottom(subm) \mid subm \in M\} \cup$
  $\{right(subm), bottom(subm), rightgrounded(subm) \mid subm \in M\}$;
$multiplicationTask_2 =$
  $\{top(subm), leftgrounded(subm), right(subm) \mid subm \in M\}$;
$multiplicationTask_3 =$
  $\{top(subm), left(subm), rightgrounded(subm) \mid subm \in M\}$;
$performMultiplication(multiplicationTask_1)$;
$completeLayer(leftsublayer(M) \cup rightsublayer(M))$;
$performMultiplication(multiplicationTask_2)$;
$performMultiplication(multiplicationTask_3)$;
$completeLayer(topsublayer(M))$

performMultiplication($tasks$):
for $(m, m1, m2) \in tasks$ do
  $P_m = P_m \cup (T_{m1} \times T_{m2})$

*Листинг 2. Модификация алгоритма Валианта*

*Listing 2. Modification of Valiant's algorithm*

Процедура $completeVLayer(M)$ принимает слой непересекающихся подматриц одинакового размера $M$. Для каждой $subm = (l, m, l', m') \in M$ эта процедура вычисляет $left(subm)$, $right(subm)$, $top(subm)$. Для корректной работы этой процедуры, во-первых, необходимо, чтобы элементы $bottom(subm)$ и $T_{i,j}$ для всех $i$ и $j$, таких что $l \le i < j \le m$ и $l' \le i < j \le m'$ уже были построены. Во-вторых, текущее значение $P_{i,j}$ для всех $i$ и $j$, таких что $l \le i < m$ и $l' \le j < m'$, должно быть следующим:

$$P_{i,j} = \{(B, C): \exists k, (m \le k < l'), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}.$$

Процедура $completeLayer(M)$ также принимает на вход набор подматриц $M$, но каждую $subm \in M$ заполняет полностью. Ограничения на входные данные такие же, как у процедуры $completeVLayer()$, за исключением условия на $bottom(subm)$, которое в данном случае не нужно.

Другими словами, $completeVLayer(M)$ вычисляет весь $V$-образный слой $M$, а $completeLayer(M_2)$ – это вспомогательная функция, необходимая для вычисления меньших квадратных подматриц слоя $M$.

И наконец процедура $performMultiplication(tasks)$, где $tasks$ – это массив троек подматриц, представляет основной шаг алгоритма: перемножение матриц. Стоит заметить, что в отличие от исходного алгоритма $|tasks| \ge 1$ и каждый $task \in tasks$ может быть выполнен параллельно.

Для представленного алгоритма справедливы следующие утверждения.

**Лемма 1.** Пусть $M$ – слой. Если для всех $(l, m, l', m') \in M$:

- для всех $i$ и $j$, таких что $l \le i < j \le m$ и $l' \le i < j \le m'$, $T_{i,j} = \{A: a_{i+1} \dots a_j \in L_G(A)\}$
- для всех $i$ и $j$, таких что $l \le i < m$ и $l' \le j < m'$, $P_{i,j} = \{(B, C): \exists k, (m \le k < l'), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}$

Тогда процедура $completeLayer(M)$ возвращает корректно заполненные $T_{i,j}$ для всех $i$ и $j$, таких что $l \le i < m$ и $l' \le j < m'$ для каждой $(l, m, l', m') \in M$.

**Доказательство.**

*По индукции по размеру матриц в слое* $(m - l)$.

**Теорема 2.** (*Корректность алгоритма*). Алгоритм из листинга 2 корректно заполняет $T_{i,j}$ для всех $i$ и $j$, и входная строка $a_1 \dots a_n \in L_G(S)$ тогда и только тогда, когда $S \in T_{0,n}$.

**Доказательство.**

Перед тем, как доказать утверждение теоремы, докажем по индукции, что все слои матрицы разбора $T$ вычисляются корректно.

*База индукции.* Слой размера $1 \times 1$ корректно заполняется в строках 2-3 листинга 2.

*Индукционный переход.* Предположим, что все слои размера $\leq 2^{p-2} \times 2^{p-2}$ вычислены корректно.

Обозначим слой размера $2^{p-1} \times 2^{p-1}$ как $M$. Будем рассматривать одну матрицу слоя $subm = (l, m, l', m')$ так, как для остальных подматриц их заполнение будет проходить аналогично.

Рассмотрим вызов процедуры $completeVLayer(M)$.

Заметим, что все $T_{i,j}$ для всех $i$ и $j$, таких что $l \leq i < j < m$ и $l' \leq i < j < m'$, уже корректно заполнены, так как эти элементы лежат в слоях, которые уже вычислены по индукционному предположению.

В начале выполнения процедуры $completeVLayer(M)$, $performMultiplication(multiplicationTask_1)$ добавляет к каждому $P_{i,j}$ все пары нетерминалов $(B, C)$, такие что $\exists k, \left(\frac{l+m}{2} \leq k < l'\right), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)$ для всех $(i, j) \in leftsublayer(M)$ и $(B, C)$, такие что $\exists k, \left(m \leq k < \frac{l'+m'}{2}\right), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)$ для всех $(i, j) \in rightsublayer(M)$. Теперь все предусловия для вызова процедуры $completeLayer(leftsublayer(M) \cup rightsublayer(M))$ выполнены и он вернет корректно заполненные $leftsublayer(M) \cup rightsublayer(M)$.

Затем процедура $performMultiplication$ вызывается с аргументами $multiplicationTask_2$ и $multiplicationTask_3$ и добавляет пары нетерминалов $(B, C)$, такие что $\exists k, \left(\frac{l+m}{2} \leq k < m\right), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)$ и $(B, C)$, такие что $\exists k, \left(l' \leq k < \frac{l'+m'}{2}\right), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)$ к каждому $P_{i,j}$ для всех $(i, j) \in topsublayer(M)$. Так как $m' = l$ (из построения слоя), условия на матрицу $P$ выполнены и $completeLayer(topsublayer)$ вернет корректно заполненный $topsublayer(M)$.

Таким образом, $completeVLayer(M)$ возвращает корректные $T_{i,j}$ для всех $(i, j) \in M$ для всех слоев $M$ матрицы $T$ и строки 4-6 листинга 2 возвращают все $T_{i,j} = \{A: A \in N, a_{i+1} \dots a_j \in L_G(A)\}$, ч. и т. д.

**Лемма 2.** Пусть $\text{calls}_i$ – это количество вызовов процедуры $completeVLayer(M)$, где для всех $(l, m, l', m') \in M$ выполнено $m - l = 2^{p-i}$.

- для всех $i \in \{1, \dots, p-1\}$ $\sum_{n=1}^{calls_i} |M| = 2^{2i-1} - 2^{i-1}$;
- для всех $i \in \{1, \dots, p-1\}$ матрицы размера $2^{p-i} \times 2^{p-i}$ перемножаются ровно $2^{2i-1} - 2^i$ раз.

**Доказательство.**

Сначала докажем первое утверждение по индукции по $i$.

*База индукции.* При $i = 1$: $calls_1 = 1$ и $|M| = 1$. Следовательно, $2^{2i-1} - 2^{i-1} = 2^1 - 2^0 = 1$.

*Индукционный переход.* Предположим, что $\sum_{n=1}^{calls_i} |M| = 2^{2i-1} - 2^{i-1}$ для всех $i \in \{1, \ldots, j\}$.

Рассмотрим $i = j + 1$.

Заметим, что функция $constructLayer(i)$ возвращает $2^{p-i} - 1$ матриц размера $2^i \times 2^i$, то есть в вызове процедуры $completeVLayer(constructLayer(k-i))$ $constructLayer(k-i)$ вернет $2^i - 1$ матриц размера $2^{p-i} \times 2^{p-i}$. Также, $completeVLayer(M)$ будет вызвано 3 раза для левых, правых и верхних подматриц матриц размера $2^{p-(i-1)} \times 2^{p-(i-1)}$. Кроме того, $completeVLayer(M)$ вызывается 4 раза для нижних, левых, правых и верхних подматриц матриц размера $2^{p-(i-2)} \times 2^{p-(i-2)}$, за исключением левых, правых и верхних подматриц матриц размера $2^{i-2} - 1$ матриц, которые к этому моменту уже были посчитаны.

Таким образом, $\sum_{n=1}^{calls_i} |M| = 2^i - 1 + 3 \times (2^{2(i-1)-1} - 2^{(i-1)-1}) + 4 \times (2^{2(i-2)-1} - 2^{(i-2)-1}) - (2^{i-2} - 1) = 2^{2i-1} - 2^{i-1}$.

Теперь мы знаем, что $\sum_{n=1}^{calls_{i-1}} |M| = 2^{2(i-1)-1} - 2^{(i-1)-1}$, и можем доказать второе утверждение: посчитаем количество перемножений матриц размера $2^{p-i} \times 2^{p-i}$. $performMultiplication$ вызывается 3 раза, $|multiplicationTask_1| = 2 \times (2^{2(i-1)-1} - 2^{(i-1)-1})$ и $|multiplicationTask_2| = |multiplicationTask_3| = 2^{2(i-1)-1} - 2^{(i-1)-1}$. То есть, количество перемножений подматриц размера $2^{p-i} \times 2^{p-i}$ равно $4 \times (2^{2(i-1)-1} - 2^{(i-1)-1}) = 2^{2i-1} - 2^i$, ч. и т. д.

**Теорема 2.** (*Оценка сложности алгоритма*). Пусть $|G|$ - длина описания грамматики $G$ и $n$ – длина входной строки. Тогда алгоритм из листинга 2 заполняет матрицу $T$ за $O(|G|BMM(n)log(n))$, где $BMM(n)$ — время, необходимое для перемножения двух булевых матриц размера $n \times n$.

**Доказательство.**

Так как в лемме 2 было показано, что количество перемножений матриц не изменилось по сравнению с исходной версией алгоритма Валианта, то доказательство будет идентично доказательству, приведенному Охотиным [8].

Таким образом, мы доказали корректность предложенной модификации, а также показали, что сложность алгоритма осталась прежней.

## 4. Применение алгоритма к задаче поиска подстрок

В данном разделе мы продемонстрируем, как модификация алгоритма Валианта может быть применена к задаче поиска подстрок.

Пусть мы хотим для входной строки размера $n = 2^p$ найти все подстроки размера $s$, которые принадлежат языку, заданному грамматикой $G$. Тогда мы должны посчитать слои подматриц, размер которых не превышает $2^r$, где $2^{r-2} < s \leq 2^{r-1}$.

Пусть $r = p - (m-2)$ и, следовательно, $(m-2) = p - r$.

Для всех $m \leq i \leq p$ количество перемножений матриц размера $2^{p-i} \times 2^{p-i}$ выполняется ровно $2^{2i-1} - 2^i$ раз и каждое из них включает перемножение $C = O(|G|)$ булевых подматриц.

$$C \cdot \sum_{i=m}^{p} 2^{2i-1} \cdot 2^{\omega(p-i)} \cdot f(2^{p-i}) = C \cdot 2^{\omega r} \cdot \sum_{i=2}^{r} 2^{(2-\omega)i} \cdot 2^{2(p-r)-1} \cdot f(2^{r-i})$$

$$\leq C \cdot 2^{\omega r} \cdot f(2^r) \cdot 2^{2(p-r)-1} \cdot \sum_{i=2}^{r} 2^{(2-\omega)i} = BMM(2^r) \cdot 2^{2(p-r)-1} \cdot \sum_{i=2}^{r} 2^{(2-\omega)i}$$

Временная сложность алгоритма для поиска всех подстрок длины $s$ равна $O(2^{2(p-r)-1}|G|BMM(2^r)log(n))$, где появившийся дополнительный множитель обозначает количество матриц в последнем вычисленном слое, но он, во-первых, мал относительно общей работы алгоритма, во-вторых, не существенен, так как эти матрицы могут быть обработаны параллельно. Алгоритм Валианта, в отличие от модификации, не может так легко быть применен к данной задаче. В нем необходимо будет полностью вычислить, как минимум, две треугольные подматрицы размера $\frac{n}{2}$ (как показано на рис. 5). Это значит, что минимальная сложность, улучшить которую без дополнительных модификаций не удастся, будет составлять $O(|G|BMM(2^{p-1})log(p-2))$.

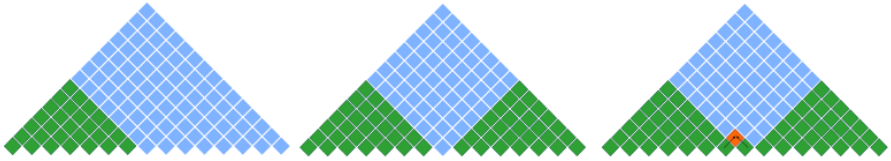

*Рис. 5. Количество элементов, которые необходимо вычислить в алгоритме Валианта (2 треугольных подматрицы размера $\frac{n}{2}$).*

*Fig. 5. The number of elements nessesary to compute in Valiant's algorithm (at least 2 triangle submatrices of size $\frac{n}{2}$).*

В завершение данного раздела, скажем, что модификация может быть эффективно применена для строк размера $s \ll n$, что и было показано в проведённых экспериментах.

## *5. Эксперименты*

В этом разделе мы приводим результаты экспериментов, целью которых является демонстрация практической применимости предложенной модификации алгоритма Валианта к задаче поиска подстрок.

### 5.1 Постановка экспериментов

Эксперименты проводились на рабочей станции со следующими характеристиками: операционная система — Linux Mint 19.1, ЦПУ — Intel i5-8250U, 1600-3400 Mhz, 4 Core(s), 8 Logical Processor(s), оперативная память — 8 GB.

Была выполнена реализация алгоритма Валианта и предложенной модификации на языке программирования C++ [14]. Также модифицированная версия была адаптирована для задачи поиска подстрок. Для перемножения подматриц использовалась библиотека для высокоэффективной работы с булевыми матрицами M4RI [9].

Сначала был проведен сравнительный анализ производительности двух версий алгоритма: исходной и модифицированной.

Затем, для адаптированной под задачу поиска подстрок версии, был проведен подсчет времени, затрачиваемого на одну строку, который позволит сделать вывод о применимости предложенной модификации в областях, работающих с большими объемами данных, например, биоинформатике.

Для экспериментов использовалась КС-грамматика $D_2$, порождающая язык Дика с двумя видами скобок, со стартовым нетерминалом S. Правила данной грамматики представлены на рис. 6.

$$S \rightarrow SS \mid (S) \mid [S] \mid \varepsilon$$

*Рис. 6. Грамматика $D_2$*

*Fig. 6. Grammar $D_2$*

Представленная грамматика была выбрана, потому что грамматики для описания правильных скобочных последовательностей часто применяются при анализе строк в биоинформатике.

Грамматика $D_2$ переводится в нормальную форму Хомского и подается на вход алгоритму со специально сгенерированными строками. Строки составлены следующим образом: заранее создается подстрока, принадлежащая языку Дика, далее в полную строку вставляется максимально возможное количество

созданных подстрок, которые можно разделить “перегородками” (терминалами, из-за которых все остальные строки, кроме вставленных будут невыводимыми в грамматике $D_2$). Строки были созданы таким образом, чтобы проверять корректность поставленных экспериментов и работу адаптированный версии модификации.

## 5.2 Результаты

Результаты сравнительного анализа алгоритма Валианта и его модификации приведены представлены в табл. 1, где $n$ – длина сгенерированной строки. Для двух реализаций представлено время работы алгоритмов в миллисекундах.

*Табл. 1. Результаты сравнительного анализа*

*Table 1. Evaluation results for comparative analysis*

| ***n*** | ***Valiant's Algorithm (ms)*** | ***Modification (ms)*** |
|---|---|---|
| ***127*** | *78* | *76* |
| ***255*** | *289* | *292* |
| ***511*** | *1212* | *1177* |
| ***1023*** | *4858* | *4779* |
| ***2047*** | *19613* | *19279* |
| ***4095*** | *78361* | *78279* |
| ***8191*** | *315677* | *315088* |

Результаты работы, адаптированной к задаче поиска подстрок модификации представлены в табл. 2, где $n$ – длина сгенерированной строки, $s$ – длина искомых подстрок. Для алгоритма Валианта и адаптированной версии модификации представлено время работы алгоритмов в миллисекундах.

*Табл. 2. Результаты работы алгоритма для задачи поиска подстрок*

*Table 2. Evaluation results for string-matching problem*

| ***s*** | ***n*** | ***Valiant's Algorithm (ms)*** | ***Modification for substr (ms)*** |
|---|---|---|---|
| ***250*** | ***1023*** | *4858* | *2996* |
| ***250*** | ***2047*** | *19613* | *6649* |
| ***510*** | ***2047*** | *19613* | *12178* |
| ***250*** | ***4095*** | *78361* | *13825* |

| | | | |
|---|---|---|---|
| ***510*** | ***4095*** | *78361* | *26576* |
| ***1020*** | ***4095*** | *78361* | *48314* |
| ***250*** | ***8191*** | *315677* | *28904* |
| ***510*** | ***8191*** | *315677* | *56703* |
| ***1020*** | ***8191*** | *315677* | *108382* |
| ***2040*** | ***8191*** | *315677* | *197324* |

## 5.3 Анализ результатов

Результаты сравнительного показывают, что предложенная модификация и исходный алгоритм Валианта работают практически одинаково.

Результаты второго эксперимента показывают, что адаптированная версия модификации может быть эффективно применена к данной задаче, она корректно находит все выводимые подстроки в строке и работает существенно быстрее алгоритма Валианта, который совершает большое количество лишних вычислений из-за сложности его преждевременной остановки.

Таким образом, поставленные эксперименты демонстрируют практическую применимость предложенной модификации алгоритма Валианта.

## *6. Заключение*

В данной работе был предложена модификация алгоритма Валианта, которая обладает некоторыми преимуществами по сравнению с исходной версией. За счет разбиения исходной матрицы на слои подматриц появилась возможность останавливать работу алгоритма, не досчитывая всю матрицу разбора до конца, если этого требует поставленная задача. Проведенные эксперименты показали, что модификация не проигрывает в производительности исходной версии алгоритма. Более того, была показана применимость данной модификации к задаче поиска подстрок.

Кроме того, мы можем определить несколько направлений будущих исследований. Так как все подматрицы в слое могут быть обработаны независимо, необходимо проверить, насколько эффективно могут быть применены техники параллельных вычислений.

Также, открытым остается вопрос, можно ли как-нибудь еще изменить порядок перемножения подматриц, чтобы полностью избавить алгоритм от рекурсивных вызовов.

### *Благодарности*

Авторы выражают признательность Кознову Дмитрию Владимировичу за оказанную помощью при написании настоящей статьи. Данная работа выполнена при финансовой поддержке гранта от JetBrains Research.

## Список литературы

[1]. Okhotin A. 2001. Conjunctive grammars // Journal of Automata, Languages and Combinatorics. 6(4), pp. 519—535.
[2]. Chomsky N. 1959. On certain formal properties of grammars // Information and control. 2(2), pp. 137—167.
[3]. Kasami T. 1965. AN EFFICIENT RECOGNITION AND SYNTAXANALYSIS ALGORITHM FOR CONTEXT-FREE LANGUAGES. Technical Report. DTIC Document.
[4]. Younger D. H. 1967. Recognition and parsing of context-free languages in time n3 // Information and control. 10(2), pp. 189—208.
[5]. Valiant L. G. 1975. General context-free recognition in less than cubic time // Journal of computer and system sciences. 10(2), pp. 308— 315.
[6]. Okhotin A. 2013. Conjunctive and Boolean grammars: the true general case of the context-free grammars // Computer Science Review. 9, pp. 27—59.
[7]. Okhotin A. 2004. Boolean grammars. Information and Computation 194, 1 (2004), pp. 19–48.
[8]. Okhotin A. 2014. Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science 516 (2014), pp. 101–120.
[9]. Albrecht, M., Bard, G. 2019. The M4RI Library. The M4RI Team. https://bitbucket.org/malb/m4ri
[10]. Earley, J. 1970. An effcient context-free parsing algorithm. Commun. ACM 13(2), pp.94-102
[11]. Bernardy, J.P., Claessen, K. 2013. Effcient divide-and-conquer parsing of practical context-free languages. SIGPLAN Not. 48(9), pp.111-122
[12]. Grigorev S., Lunina P. The Composition of Dense Neural Networks and Formal Grammars for Secondary Structure Analysis.
[13]. Rivas, E., & Eddy, S. R. 2000. The language of RNA: a formal grammar that includes pseudoknots. Bioinformatics, 16(4), pp. 334-340.
[14]. Stroustrup, B. (2000). The C++ programming language. Pearson Education India.

# Modification of Valiant's algorithm for the string-matching problem

[1] *Y.A. Susanina* <jsusanina@gmail.com>
[3] *A.N. Yaveyn* <yaveyn@yandex.ru>
[3] *S.V. Grigorev* <Semen.Grigorev@jetbrains.com>
[1, 2] *Saint Petersburg State University,*
*7/9, Universitetskaya nab., St. Petersburg, 199034, Russia.*

**Abstract.**

The theory of formal languages and, particularly, context-free grammars has been extensively studied and applied in different areas. For example, several approaches to the recognition and classification problems in bioinformatics are based on searching the genomic subsequences possessing some specific features which can be described by a context-free grammar. Therefore, the string-matching problem can be reduced to parsing – verification if some subsequence can be derived in this grammar. Such field of application as bioinformatics requires working with a large amount of data, so it is necessary to improve the existing parsing techniques. The most asymptotically efficient parsing algorithm that can be applied to any context-free grammar is a matrix-based algorithm proposed by Valiant. This paper aims to present Valiant's algorithm modification, which main advantage is the possibility to divide the parsing table into successively computed layers of disjoint submatrices where each submatrix of the layer can be processed independently. Moreover, our approach is easily adapted for the string-matching problem. Our evaluation shows that the proposed modification retains all benefits of Valiant's algorithm, especially its high performance achieved by using fast matrix multiplication methods. Also, the modified version decreases a large amount of excessive computations and accelerates the substrings searching.

**Keywords:** parsing, context-free grammars, matrix operations.

**DOI:** ???

**For citation:** Susanina Y.A., Yaveyn A.N., Grigorev S.V. Title. *Trudy ISP RAN/Proc. ISP RAS*, vol. ?, issue ?, 2019. pp. ?-? (in Russian). DOI: ???

## References

[1]. Okhotin A. 2001. Conjunctive grammars // Journal of Automata, Languages and Combinatorics. 6(4), pp. 519—535.
[2]. Chomsky N. 1959. On certain formal properties of grammars // Information and control. 2(2), pp. 137—167.

[3]. Kasami T. 1965. AN EFFICIENT RECOGNITION AND SYNTAX ANALYSIS ALGORITHM FOR CONTEXT-FREE LANGUAGES. Technical Report. DTIC Document.
[4]. Younger D. H. 1967. Recognition and parsing of context-free languages in time n3 // Information and control. 10(2), pp. 189—208.
[5]. Valiant L. G. 1975. General context-free recognition in less than cubic time // Journal of computer and system sciences. 10(2), pp. 308— 315.
[6]. Okhotin A. 2013. Conjunctive and Boolean grammars: the true general case of the context-free grammars // Computer Science Review. 9, pp. 27—59.
[7]. Okhotin A. 2004. Boolean grammars. Information and Computation 194, 1 (2004), pp. 19–48.
[8]. Okhotin A. 2014. Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science 516 (2014), pp. 101–120.
[9]. Albrecht, M., Bard, G. 2019. The M4RI Library. The M4RI Team. https://bitbucket.org/malb/m4ri
[10]. Earley, J. 1970. An effcient context-free parsing algorithm. Commun. ACM 13(2), pp.94-102
[11]. Bernardy, J.P., Claessen, K. 2013. Effcient divide-and-conquer parsing of practical context-free languages. SIGPLAN Not. 48(9), pp.111-122
[12]. Grigorev S., Lunina P. The Composition of Dense Neural Networks and Formal Grammars for Secondary Structure Analysis.
[13]. Rivas, E., & Eddy, S. R. 2000. The language of RNA: a formal grammar that includes pseudoknots. Bioinformatics, 16(4), pp. 334-340.
[14]. Stroustrup, B. (2000). The C++ programming language. Pearson Education India.