Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.М07-мм

# Реализация матричного алгоритма контекстно-свободной достижимости на Java

***СЧЕРЕВСКИЙ Виктор Максимович***

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
инженер-исследователь, лаборатория технологий программирования, Кутуев В.А.

Санкт-Петербург
2023

# Оглавление

# Введение

В теории графов одной из популярных задач является поиск путей между вершинами графа. В некоторых формулировках этой задачи на искомые пути могут накладываться какие-либо ограничения. Например, при работе с графом, рёбра которого помечены символами некоторого алфавита, можно потребовать, чтобы слова, формируемые путями, принадлежали заданному на этом алфавите формальному языку. Такой класс задач называется задачами поиска путей с ограничениями в терминах формальных языков, а в случае контекстно-свободных (КС) языков, – задачами КС-достижимости [13].

Подобного рода задачи часто встречаются в областях, связанных с анализом граф-структурированных данных, например, при статическом анализе кода, в графовых базах данных и в биоинформатике [13, 2].

Одним из возможных алгоритмов решения задачи КС-достижимости является матричный алгоритм, предложенный в статье Рустама Азимова и Семёна Григорьева [2]. В данной работе предлагается реализовать этот алгоритм на языке Java с использованием возможностей существующих библиотек матричной алгебры.

# 1. Постановка задачи

Целью данной работы является реализация матричного алгоритма КС-достижимости на Java.

Для её выполнения были поставлены следующие задачи:

1. Провести обзор существующих решений на платформе Java, которые могут быть полезны при решении задач поиска путей с ограничениями;
2. Реализовать рабочий прототип матричного алгоритма КС-достижимости на Java с использованием возможностей существующих библиотек;
3. Провести экспериментальный анализ полученного решения.

# 2. Обзор

В данном разделе будут даны основные понятия из теории формальных языков и теории графов, необходимые для понимания задачи. Далее приведён краткий анализ матричного алгоритма КС-достижимости, а также обзор библиотек для JVM приложений, на основе которых можно этот алгоритм реализовать.

## 2.1. Основные определения

**Контекстно-свободная грамматика** - $G = (\Sigma, N, P, S)$, где:

- $\Sigma$ – множество терминальных символов;
- $N$ – множество нетерминальных символов;
- $P$ – множество продукций (правил вывода), каждая из которых имеет вид $A \to \alpha$, где $A \in N, \alpha \in (\Sigma \cup N)^*$;
- $S \in N$ – стартовый нетерминал.

Контекстно-свободная грамматика находится в **ослабленной нормальной форме Хомского**, если каждая продукция имеет вид одной из следующих:

- $A \to BC$, где $A, B, C \in N$;
- $A \to a$ , где $A \in N, a \in \Sigma$;
- $A \to \varepsilon$, где $A \in N$, $\varepsilon$ – пустой символ.

**Помеченный ориентированный граф (с метками на рёбрах)** – это граф $G = (V, E, L)$, где $V$ – множество вершин, $E = V \times L \times V$ – множество рёбер и $L$ – множество меток графа. Будем считать, что вершины графа индексируются целыми числами, начиная с 0, т.е. $V = \{0, ..., |V| - 1\}$.

**Путь** $\pi$ в графе $G = (V, E, L)$ – это последовательность рёбер $e_0, e_1, ..., e_{n-1}$, где $e_i = (v_i, l_i, u_i) \in E$. Путь между вершинами $v$ и $u$ обозначим как $v\pi u$. Путь $\pi = (v_0, l_0, v_1), ..., (v_{n-1}, j_{n-1}, v_n)$ формирует слово $\omega(\pi) = l_0...l_{n-1}$.

Пусть контекстно-свободная грамматика формирует язык $\mathcal{L}$. Тогда задача КС-достижимости заключается в поиске всех таких пар вершин $(v, u)$, что между ними существует путь $\pi$ такой, что $\omega(\pi) \in \mathcal{L}$.

## 2.2. Анализ алгоритма

Матричный алгоритм КС-достижимости использует идею транзитивного замыкания графа, т.е. получение такого графа с исходным множеством вершин, у которого между каждой парой вершин есть ребро тогда и только тогда, когда в исходном графе между этими вершинами существует путь. В нашем случае путь должен формировать слово из контекстно-свободного языка, а рёбра транзитивного замыкания графа помечены множеством нетерминалов, из которых выводится слово, соответствующее некоторому пути. Поскольку множество нетерминалов конечно, для каждого из них можно выделить булеву матрицу, которая идентифицирует наличие путей, выводимых из соответствующего нетерминала, для каждой пары вершин. Псевдокод алгоритма представлен ниже [13].

**Listing 1** Context-free path querying algorithm. Boolean matrix version

```
1: function evalCFPQ(D = (V, E), G = (N, Σ, P))
2:     n ← |V|
3:     T ← {T^{A_i} | A_i ∈ N, T^{A_i} is a matrix n × n, T^{A_i}_{k,l} ← false}
4:     for all (i, x, j) ∈ E, A_k | A_k → x ∈ P do T^{A_k}_{i,j} ← true
5:     for A_k | A_k → ε ∈ P do T^{A_k}_{i,i} ← true
6:     while any matrix in T is changing do
7:         for A_i → A_j A_k ∈ P do T^{A_i} ← T^{A_i} + (T^{A_j} × T^{A_k})
       return T
```

Как можно видеть, алгоритм состоит из трёх частей: инициализация матрицы терминальными выводами (вида $A \to x$), добавление петель

(для правил вида $A \to \varepsilon$) и вычисление транзитивного замыкания для правил вида $A \to BC$. Других продукций быть не должно, т.е. предполагается, что грамматика находится в ослабленной нормальной форме Хомского.

Для реализации этого алгоритма необходимо иметь 3 структуры: грамматику в ослабленной нормальной форме Хомского, граф, представленный списком рёбер, и булеву матрицу, поддерживающую операции сложения и умножения. Это базовая, довольно типичная для многих задач функциональность, поэтому велика вероятность того, что она уже реализована в каких-либо библиотеках для JVM приложений.

## 2.3. Обзор существующих решений

Поиск подходящих библиотек осуществлялся по ключевым словам ("Java matrix library", "Java linear algebra", "JVM matrix" и др.) среди open-source проектов на GitHub, в глобальных репозиториях Maven [10] и с помощью различных поисковых систем в интернете. Каждая найденная библиотека проверялась на соответствие поставленным требованиям по её документации. Если в документации явным образом не обозначено наличие или отсутствие искомой функциональности, поиск проводился по исходному коду библиотеки, при его доступности, и по сигнатуре отдельных классов и интерфейсов, в противном случае.

В процессе поиска существующих решений для указанных проблем не было найдено библиотек в открытом доступе для работы с формальными языками на Java. Для работы с графами нет необходимости искать библиотеку с высокопроизводительными алгоритмами, достаточно реализовать простейший формат хранения. А для матричных вычислений, на которые приходится основная алгоритмическая нагрузка, было найдено несколько библиотек, но ни одна из них в полной мере не предоставляет необходимую функциональность. Далее приведено краткое описание некоторых из них.

Первая из рассмотренных библиотек - EJML [6]. Это одна из наиболее популярных библиотек линейной алгебры (2 место в категории

“Vector/Matrix Libraries” рейтинга MvnRepository [12]). Однако выяснилось, что поддержка булевых матриц в ней сильно ограничена (нет операций сложения и умножения), а целочисленных матриц нет вообще [7].

Возможным вариантом оказалась библиотека Colt [5], поскольку в ней есть битовые вектора и матрицы (BitMatrix). Но операции сложения и умножения на них также не реализованы, хотя их можно выразить через другие логические операции. Также есть матрица произвольных объектов ObjectMatrix2D, но необходимых операций на ней тоже нет.

Другой вариант – библиотека Apache Commons Math [1]. Она предоставляет типизируемый интерфейс FieldMatrix, в котором определены все необходимые операции, а также различные его реализации, в том числе в разреженном формате. Для использования этих классов необходимо реализовать интерфейс FieldElement (скалярные операции сложения, умножения и некоторые другие), который инкапсулирует тип элемента матрицы. Было принято решение выбрать именно эту библиотеку.

# 3. Реализация проекта

Для решения поставленной задачи был реализован полноценный проект, включающий саму реализацию алгоритма, необходимую инфраструктуру, а также инструменты для анализа решения. Проект был назван CFPQ_JavaAlgo, исходный код доступен в репозитории GitHub [4]. В данном разделе описан процесс разработки проекта.

## 3.1. Подготовка инфраструктуры

Прежде чем приступить к непосредственной реализации алгоритма, была подготовлена необходимая инфраструктура. Класс графа EdgeListGraph представляет собой обыкновенный список рёбер. Реализована возможность чтения графа из файла в триплет-формате (каждая строка – тройка (вершина, метка ребра, вершина)).

Для представления грамматики был реализован набор классов: терминал, нетерминал, пустой символ, базовый символ, слово, продукция, абстрактная грамматика и непосредственно КС-грамматика – ContextFreeGrammar. Отдельно отмечу способ хранения символов и продукций. Они хранятся в хэш-множестве (HashSet), а в классах примитивов переопределены методы hashCode и equals, благодаря чему исключается возможность дублирования символов и продукций грамматики. Для КС-грамматики также была реализована функциональность чтения из файла.

Для возможности работы с булевыми матрицами были реализованы интерфейсы Field (нулевой и единичный элементы алгебраической структуры) и FieldElement (элемент алгебраической структуры) библиотеки Apache Commons Math. В основе лежит примитив Boolean, соответствующие классы: BoolField и BoolFieldElement – частичная реализация FieldElement с операциями сложения и умножения (т.е. полученные классы образуют полукольцо). Далее появилась возможность типизировать классы SparseFieldMatrix и Array2DRowFieldMatrix (BoolSparseMatrix и BoolDenseMatrix соответственно). Для дальнейшей реализации алгоритма используется разреженный вариант.

## 3.2. Реализация алгоритма

Алгоритм реализован в точности, как предложено его авторами, включая одну из оптимизаций [11]. Перед инициализацией матриц продукции грамматики разделяются на 3 коллекции: продукции вида $A \to x$, $A \to \varepsilon$ и $A \to BC$. Благодаря этому сокращается количество итераций в каждом цикле.

Наиболее трудоемкая часть алгоритма находится в цикле, где происходит умножение и сложение булевых матриц. Насчет оптимальности этих операций пока нельзя ничего сказать, поскольку библиотекой явно не предусматривается оптимизация для булевых значений. Тем не менее, положительно на производительность алгоритма может повлиять разреженный формат хранения матриц или параллельное выполнение матричных операций, если оно предусмотрено библиотекой.

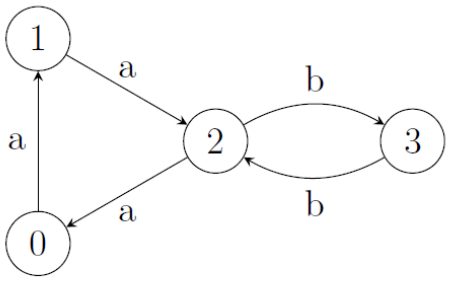
# 4. Анализ полученного решения

## 4.1. Проверка корректности алгоритма

Для проверки работоспособности алгоритма были реализованы юнит-тесты, проверяющие корректность реализаций как примитивов – графа, грамматики и булевой матрицы, так и самого алгоритма КС-достижимости. Для тестирования используется фреймворк JUnit 5 [9].

Один из тестов запускает алгоритм на графе и грамматике небольшого размера (рис.1) [13] и сравнивает результат поэлементно для каждой матрицы достижимости (для каждого нетерминала). Остальные тесты алгоритма сравнивают количество достижимых пар из стартового нетерминала.

граф $\mathcal{G}$:




грамматика $G$:

$$S \to AB \qquad A \to a$$
$$S \to AS_1 \qquad B \to b$$
$$S_1 \to SB$$

Рис. 1: юнит-тест

## 4.2. Анализ производительности

Для измерения производительности алгоритма использовался фреймворк JMH [8]. Был реализован бенчмарк, который запускает алгоритм на данных из датасета проекта CFPQ-on-GPGPU [3]

(WorstCase, худший случай). Запуск происходил на матрицах графов WorstCaseMatrix_3 , WorstCaseMatrix_4, WorstCaseMatrix_5, WorstCaseMatrix_6 и WorstCaseMatrix_7, каждая из которых характеризуется своим размером. Измерение каждого случая производилось 20 раз, после чего результат усреднялся.

Программно-аппаратная конфигурация приведена в таблице ниже:

| | |
|---|---|
| Процессор | Intel Core i7-8750H CPU 2.20GHz (Coffee Lake), 1 CPU, 12 logical and 6 physical cores |
| Операционная система | Windows 11 23H2 |
| Версия Java | 17.0.1 |
| Версия JMH | 1.37 |

Таблица 1: конфигурация

Результаты измерений (среднее ± стандартное отклонение):

| WorstCaseMatrix_ | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|
| Размер (кол-во рёбер) | 17 | 33 | 65 | 129 | 257 |
| Время (мс) | 21.27 ± 0.13 | 670 ± 12 | 20662 ± 290 | 622447 ± 4489 | $> 7.2 \cdot 10^6$ |

Таблица 2: результаты измерений

Как можно видеть, время работы алгоритма растёт очень быстро относительно размера графа. В случае WorstCaseMatrix_7 выполнение шло слишком долго (более двух часов), поэтому бенчмарк был остановлен. Судить об эффективности алгоритма еще рано, т.к. был исследован только худший случай и одна реализация. Тем не менее, стало ясно, что для задач большой размерности практической пользы от полученного решения мало, поэтому следует искать более оптимальный подход.

# Заключение

В ходе работы были достигнуты следующие результаты:

- Сформулированы требования к библиотекам для работы с примитивами;
- Проведён обзор существующих библиотек для матричных вычислений на Java. Библиотек для работы с формальными языками на Java не было обнаружено. Также было выяснено, что библиотеки линейной алгебры в большинстве своём не поддерживают вычисления на целочисленных и булевых матрицах, либо эта поддержка сильно ограничена;
- Реализована первая версия матричного алгоритма КС-достижимости с использованием одной из библиотек;
- Проведён экспериментальный анализ полученного решения. Корректность работы алгоритма подтвердилась тестами, а производительность была измерена для набора данных в худшем случае.

Дальнейшее исследование может быть направлено на реализацию других алгоритмов КС-достижимости, оптимизацию текущей версии матричного алгоритма КС-достижимости, а также на разработку средств целочисленной и булевой матричной алгебры и средств для работы с формальными языками на Java.

# Список литературы


[1] Apache Commons Math. — URL: https://commons.apache.org/proper/commons-math/ (дата обращения: 28 декабря 2023 г.).

[2] Azimov Rustam, Grigorev Semyon. Context-Free Path Querying by Matrix Multiplication // Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). — GRADES-NDA '18. — New York, NY, USA : Association for Computing Machinery, 2018. — 10 p. — URL: https://doi.org/10.1145/3210259.3210264.

[3] CFPQ-on-GPGPU. — URL: https://github.com/JetBrains-Research/CFPQ-on-GPGPU (дата обращения: 28 декабря 2023 г.).

[4] CFPQ_JavaAlgo. — URL: https://github.com/kolbacer/CFPQ_JavaAlgo (дата обращения: 28 декабря 2023 г.).

[5] Colt. — URL: https://dst.lbl.gov/ACSSoftware/colt/ (дата обращения: 28 декабря 2023 г.).

[6] EJML. — URL: http://ejml.org/wiki/index.php (дата обращения: 28 декабря 2023 г.).

[7] EJML issue 68. — URL: https://github.com/lessthanoptimal/ejml/issues/68 (дата обращения: 28 декабря 2023 г.).

[8] JMH. — URL: https://github.com/openjdk/jmh (дата обращения: 28 декабря 2023 г.).

[9] JUnit 5. — URL: https://junit.org/junit5/ (дата обращения: 28 декабря 2023 г.).

[10] Maven repositories. — URL: https://mvnrepository.com/repos (дата обращения: 28 декабря 2023 г.).

[11] Muravjov Ilya. Optimization of the Context-Free Language Reachability Matrix-Based Algorithm.

[12] MvnRepository "Vector/Matrix Libraries" rating. — URL: https://mvnrepository.com/open-source/vector-and-matrix-libraries (дата обращения: 28 декабря 2023 г.).

[13] Григорьев С. В. О достижимости с ограничениями в терминах формальных языков. — URL: https://github.com/FormalLanguageConstrainedPathQuerying/FormalLanguageConstrainedReachability-LectureNotes (дата обращения: 28 декабря 2023 г.).