САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

На правах рукописи

Азимов Рустам Шухратуллович

Решение задач поиска путей в графе с заданными контекстно-свободными ограничениями с использованием методов линейной алгебры

Научная специальность
2.3.5. Математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ
на соискание учёной степени кандидата физико-математических наук

Научный руководитель:
кандидат физико-математических наук
Григорьев Семён Вячеславович

Санкт-Петербург
2022

Оглавление

Стр.

Стр.

Введение

В современном мире становится всё больше данных, которые требуют обработки и анализа. При этом графы являются одной из самых распространённых и удобных структур, позволяя компактно представлять большие объёмы информации и реализовывать эффективные алгоритмы для её анализа. Графы используются в статическом анализе программ [1; 2] и биоинформатике [3], в социальных сетях [4], сетевом анализе [5] и т.д. Также в настоящее время активно развиваются графовые базы данных, используемые для хранения данных в виде графов и реализации запросов к ним. Следует упомянуть такие графовые базы данных, как RedisGraph[1] и Neo4j[2].

Одной из важнейших задач анализа графов является поиск путей. Имеются разные вариации этой задачи: непосредственный поиск определённых путей, задача достижимости (сами пути не ищутся, но доказывается их существование) и т.д.

В рамках этих задач могут задаваться специальные свойства искомых путей. Одним из способов описания таких свойств является использование формального языка над некоторым алфавитом [6]. Таким способом можно ограничить множество слов, получаемых конкатенацией меток на дугах рассматриваемых путей. Таким образом, в задачах поиска путей в помеченном графе с заданным формальным языком рассматриваются только те пути, которые образуют слова, принадлежащие этому языку. В настоящее время активно исследуются ограничения, представленные в виде контекстно-свободных (КС) языков [1; 7—10]. Такие языки позволяют описывать более широкий набор ограничений, чем активно используемые на практике регулярные выражения [11; 12].

С практической точки зрения, одним из распространённых способов получения высокопроизводительных реализаций алгоритмов анализа графов является использование методов линейной алгебры [13]. При этом существующие алгоритмы, фактически, переводятся на язык линейной алгебры, т.е. для представления графа используются разреженные матрицы (такие матри-

[1]Графовая база данных RedisGraph: https://oss.redislabs.com/redisgraph/ (дата обращения: 14.01.2022).

[2]Графовая база данных Neo4j: https://neo4j.com/ (дата обращения: 14.01.2022).

цы, которые имеют малое количество ненулевых элементов), а для анализа и преобразований графа используются операции над матрицами (умножение матриц, сложение, транспонирование матриц и т.д.). Например, давно известны такие представления графа, как матрица смежности или матрица инцидентности, а такому преобразованию ориентированного графа, как инвертирование направлений дуг соответствует операция транспонирования матрицы смежности. Для тех алгоритмов анализа графов, которые позволяют такой «перевод», становится возможным использовать параллельные вычисления, в частности, на основе GPU-технологий, что позволяет существенно улучшить их производительность. Кроме того, такого рода алгоритмы зачастую просты в реализации, так как позволяют использовать существующие библиотеки линейной алгебры (SuiteSparse:GraphBLAS[3], cuSPARSE[4], cuBLAS[5], cuBool[6], m4ri[7], Scipy[8] и др.).

Однако возможность использования методов линейной алгебры в задачах поиска путей в графе с заданными КС-ограничениями в настоящее время не исследована. Как показывает практика, существующие решения данной задачи страдают от недостаточной производительности и не справляются с постоянно растущими размерами реальных графов [14]. В то же время создание новых решений, использующих методы линейной алгебры, позволит решить данную проблему с помощью теоретических и практических достижений линейной алгебры.

В последнее время появилось значительное число работ, посвященных классическим алгоритмам анализа графов, переведённых на язык линейной алгебры. Например, Айдын Булук (Aydin Buluç), Упасана Шридхар (Upasana Sridhar), Питер Чжан (Peter Zhang), Арифул Азад (Ariful Azad) и Лейуань Ван

[3]SuiteSparse:GraphBLAS — реализация стандарта GraphBLAS на языке `C`: https://github.com/DrTimothyAldenDavis/GraphBLAS (дата обращения: 14.01.2022).

[4]Библиотека линейной алгебры cuSPARSE, используемая для работы с разреженными матрицами на GPU: https://docs.nvidia.com/cuda/cusparse/index.html (дата обращения: 14.01.2022).

[5]Библиотека линейной алгебры cuBLAS на основе GPU-технологий: https://docs.nvidia.com/cuda/cublas/index.html (дата обращения: 14.01.2022).

[6]Библиотека булевой линейной алгебры cuBool, используемая для работы с булевыми разреженными матрицами и векторами на GPU: https://github.com/JetBrains-Research/cuBool (дата обращения: 14.01.2022).

[7]m4ri — библиотека для быстрой арифметики с плотными булевыми матрицами: https://github.com/malb/m4ri (дата обращения: 14.01.2022).

[8]Scipy — библиотека для языка программирования `Python` с открытым исходным кодом, предназначенная для выполнения научных и инженерных расчётов: https://scipy.org/ (дата обращения: 14.01.2022).

(Leyuan Wang) в своих работах [15—19] показывают применимость на практике алгебраических версий таких алгоритмов, как поиск в ширину, алгоритм Дейкстры, алгоритм Беллмана-Форда, алгоритм поиска наибольшего паросочетания в двудольном графе и алгоритм подсчёта количества треугольников в графе.

На фоне роста популярности идеи решения задач анализа графов с помощью методов линейной алгебры относительно недавно был создан стандарт GraphBLAS [20], который определяет базовые «строительные блоки» алгоритмов анализа графов в терминах линейной алгебры. Такими «блоками», например, являются умножение и другие операции над матрицами, так как стандарт GraphBLAS использует представление графов в виде матриц смежности. Также, ввиду того, что данные на практике разрежены, целесообразно использовать разреженный формат для этих матриц. Стоит отметить, что не каждый алгоритм анализа графов можно переформулировать на языке линейной алгебры. Так, например, до сих пор это не сделано для алгоритма обхода произвольного графа, использующего поиск в глубину [21]. Также в настоящее время это не сделано и для алгоритмов поиска путей в графе с заданными КС-ограничениями.

Задача поиска путей в графе с заданными КС-ограничениями является одной из важных задач анализа графов. Её частным случаем является задача синтаксического анализа КС-языков, в которой анализируются строки, что эквивалентно анализу только линейно-помеченных графов. Лесли Вэлиант (Leslie Valiant) провёл исследование [22], посвященное синтаксическому анализу КС-языков с использованием операций над матрицами. Предложенный им субкубический алгоритм для заданных строки и КС-грамматики определяет порождается ли эта строка заданной грамматикой с помощью операций умножения булевых матриц. Впервые вопрос о возможности нахождения матричного алгоритма поиска путей в графе с заданными КС-ограничениями исследовал Михалис Яннакакис (Mihalis Yannakakis) [23]. Он указывал, что алгоритм Вэлианта может быть расширен для анализа графов без циклов, но сомневался в возможности создания субкубического алгоритма для анализа произвольных графов.

Однако для частного случая КС-ограничений существует алгоритм поиска путей в графе, сформулированный на языке линейной алгебры. Такой алгоритм был предложен Филипом Брэдфордом (Philip Bradford) [8], исследовавшим задачу достижимости в графе с заданными КС-ограничениями.

Кроме того, существует ряд работ [9; 24—26], посвященных задаче поиска путей в произвольном графе с заданными произвольными КС-ограничениями и основанных на различных алгоритмах синтаксического анализа (LR, LL, GLL, CYK). Среди них работы Семёна Григорьева, Джелле Хеллингса (Jelle Hellings), Чиро Медейроса (Ciro Medeiros) и Мартина Мюзиканте (Martin Musicante). Стоит отметить, что большинство из представленных алгоритмов требуют представить КС-ограничения на пути в графе в виде КС-грамматики в некоторой нормальной форме. Отдельный интерес представляют собой алгоритмы, не требующие дополнительных преобразований структур, описывающих входные КС-ограничения, так как почти любое такое преобразование обычно приводит к увеличению размеров этих структур, что может негативно сказаться на производительности. Кроме того, после таких преобразований могут возникнуть сложности с интерпретацией результатов анализа графа в терминах изначальной структуры, заданной пользователем. Примерами алгоритмов поиска путей в графе с заданными КС-ограничениями, не требующих преобразований входной КС-грамматики, являются алгоритмы [24; 26], основанные на алгоритмах синтаксического анализа LL и GLL.

Таким образом, на текущий момент не существует алгоритма поиска путей в произвольном графе с заданными произвольными КС-ограничениями, выраженного на языке линейной алгебры. Поэтому необходимо исследовать возможность разработки таких алгоритмов.

Целью данной работы является исследование применимости методов линейной алгебры к задаче поиска путей в графе с заданными КС-ограничениями для получения высокопроизводительных реализаций на основе параллельных вычислений.

Достижение поставленной цели обеспечивается решением следующих **задач**.

1. Разработать подход к поиску путей в графе с КС-ограничениями на основе методов линейной алгебры.
2. Разработать алгоритм, использующий предложенный подход и решающий задачи поиска путей в графе с заданными КС-ограничениями.
3. Разработать алгоритм поиска путей в графе с заданными КС-ограничениями, использующий предложенный подход и не требующий преобразования входной КС-грамматики.

4. Реализовать предложенные алгоритмы с использованием параллельных вычислений, провести их экспериментальное исследование на реальных данных, сравнить их с существующими реализациями, а также между собой.

Теоретическая и практическая значимость работы. Теоретическая значимость диссертационного исследования заключается в разработке подхода к поиску путей в графе с заданными КС-ограничениями, использующего методы линейной алгебры, в разработке формальных алгоритмов, использующих полученный подход, а также в формальном доказательстве завершаемости, корректности и оценок временной сложности разработанных алгоритмов.

В ходе исследования предложенные алгоритмы реализованы с использованием параллельных вычислений, что позволило ускорить время анализа графов до 3 порядков и уменьшить объём потребляемой памяти до 2 порядков по сравнению с существующими реализациями. Кроме того, выполненные реализации могут быть интегрированы с такими графовыми базами данных, как RedisGraph. Это позволит расширить языки запросов к этим базам данных.

Методология и методы исследования. Методология исследования основана на линейной алгебре и теории графов. В работе использован стандарт GraphBLAS, объединяющий теорию графов и линейную алгебру. Кроме того, в исследовании использовалась теория формальных языков, а также теория сложности. Наконец, для реализации алгоритмов использовались CPU и GPU-технологии.

Основные положения, выносимые на защиту.

1. Разработан подход к поиску путей в графе с заданными КС-ограничениями на основе методов линейной алгебры, который позволяет использовать теоретические и практические достижения линейной алгебры для решения данной задачи.
2. Разработан алгоритм, использующий предложенный подход и решающий задачи поиска путей в графе с заданными КС-ограничениями. Доказана завершаемость и корректность предложенного алгоритма. Получена теоретическая оценка сверху временной сложности алгоритма. Предложенный алгоритм использует операции над матрицами, которые позволяют применять широкий класс оптимизаций и дают возможность автоматически распараллеливать вычисления за счёт существующих библиотек линейной алгебры.

3. Разработан алгоритм поиска путей в графе с заданными КС-ограничениями, использующий предложенный подход и не требующий преобразования входной КС-грамматики. Доказана завершаемость и корректность предложенного алгоритма. Получена теоретическая оценка сверху временной сложности алгоритма. Предложенный алгоритм позволяет работать с произвольными входными КС-грамматиками без необходимости их преобразования, что позволяет избежать значительного увеличения размеров входной грамматики и увлечения времени работы алгоритма.
4. Предложенные алгоритмы реализованы с использованием параллельных вычислений. Проведено экспериментальное исследование разработанных алгоритмов на реальных RDF данных и графах, построенных для статического анализа программ. Было проведено сравнение полученных реализаций между собой, с существующими решениями из области статического анализа и с решениями, основанными на различных алгоритмах синтаксического анализа. Результаты сравнения показывают, что предложенные реализации для задачи достижимости позволяют ускорить время анализа до 2 порядков и потребляют до 2 раз меньше памяти по сравнению с существующими решениями, а для задач поиска одного и поиска всех путей в графе позволяют ускорить время анализа до 3 порядков и до 2 порядков снизить потребление памяти.

Научная новизна.

1. Предложен новый подход к поиску путей в графе с заданными КС-ограничениями, который позволяет использовать теоретические и практические аспекты линейной алгебры для решения задачи поиска путей в графе с КС-ограничениями.
2. Впервые получен алгоритм поиска путей в произвольном графе с заданными произвольными КС-ограничениями, сформулированный в терминах линейной алгебры, что позволяет применять в этом алгоритме широкий класс оптимизаций для вычисления операций над матрицами, распараллеливать вычисления и существенно улучшить производительность.
3. В диссертации также предложен алгоритм поиска путей в графе с заданными КС-ограничениями, сформулированный в терминах линейной

алгебры и не требующий преобразования входной КС-грамматики, в отличие от алгоритмов, предложенных в работах Семёна Григорьева, Джелле Хеллингса и Филиппа Брэдфорда. Таким образом, предложенный в диссертации алгоритм позволяет избежать значительного увеличения размера входной грамматики, от которого напрямую зависит время его работы.

Степень достоверности и апробация результатов. Достоверность и обоснованность результатов исследования опирается на использовании формальных методов доказательств и инженерные эксперименты.

Основные результаты работы были представлены на ряде международных научных конференций: Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (совместно с конференцией SIGMOD) 2018 (Хьюстон, Техас, США), 2020 (Портленд, Орегон, США) и 2021 (Сиань, Шэньси, Китай); 24th European Conference on Advances in Databases and Information Systems (ADBIS) 2020 (Лион, Франция); VLDB PhD Workshop совместно с 47th International Conference on Very Large Data Bases 2021 (Копенгаген, Дания). Также исследование было поддержано грантом РНФ №18-11-00100 и грантом РФФИ №19-37-90101.

Публикации. Все результаты диссертации изложены в 8 научных работах [27—34]. Из них 3 работы [32—34] опубликованы в журналах из «Перечня российских рецензируемых научных журналов, в которых должны быть опубликованы основные научные результаты диссертаций на соискание ученых степеней доктора и кандидата наук», рекомендовано ВАК. 7 работ [27—32; 34] индексируются в базе данных Scopus. Работы [27—30; 32—34] написаны в соавторстве. В работе [27] автору принадлежит разработка и реализация алгоритма, решающего задачу достижимости в графе с заданными КС-ограничениями с использованием методов линейной алгебры, доказательство корректности разработанного алгоритма, постановка экспериментов; соавторы участвовали в обсуждении основных идей статьи, выполняли обзор предметной области. В работе [28] автору принадлежит разработка и реализация алгоритма, решающего задачу поиска одного пути в графе с заданными КС-ограничениями с использованием методов линейной алгебры, доказательство корректности разработанного алгоритма; соавторы проводили экспериментальное исследование, участвовали в формализации и улучшении изложения идей статьи. В работе [29] вклад автора заключается в доказательстве корректности алгоритма

поиска путей в графе с заданными КС-ограничениями, не требующего преобразований входной КС-грамматики, а также в работе над текстом; соавторам принадлежит идея алгоритма и постановка экспериментов. В работе [30] автору принадлежит разработка и реализация алгоритма, решающего задачу поиска всех путей в графе с заданными КС-ограничениями с использованием методов линейной алгебры, доказательство корректности разработанного алгоритма, работа над текстом; соавторы проводили экспериментальное исследование. В работах [32; 33] вклад автора заключается в разработке и реализации алгоритма, решающего задачу достижимости в графе с заданными ограничениями в виде конъюнктивных языков, доказательстве корректности разработанного алгоритма и постановке экспериментов; соавторы участвовали в обсуждении основных идей статьи и выполняли обзор предметной области. В работе [34] автору принадлежит разработка и реализация алгоритма, решающего задачу поиска всех путей в графе с заданными КС-ограничениями с использованием матриц с множествами промежуточных вершин, а также работа над текстом; соавторы проводили экспериментальное исследование.

Благодарности. Прежде всего я бы хотел поблагодарить своего научного руководителя, Семёна Вячеславовича Григорьева, за руководство на всех этапах данного исследования, за готовность поддержать и поделиться своим опытом и за неоценимый вклад в мою работу. Также хочу поблагодарить Дмитрия Владимировича Кознова за его мудрость, активное участие и за многочисленные беседы о моей диссертации, которые оказали огромное влияние на мою работу и на меня в целом.

Я выражаю благодарность Андрею Николаевичу Терехову и кафедре системного программирования СПбГУ, а также Дмитрию Юрьевичу Булычеву, Андрею Владимировичу Иванову и компаниям JetBrains и Huawei за уникальную возможность заниматься наукой как основной деятельностью.

Я благодарен Владимиру Александровичу Кутуеву и Владе Владимировне Погожельской за помощь в постановке экспериментов.

Отдельную благодарность хочется выразить моей жене, Светлане Дмитриевне Азимовой, и моему сыну, Артёму Рустамовичу Азимову, за вдохновение, любовь и поддержку, ведь они занимают все мои мысли и всё моё сердце, и всё в этой жизни я делаю именно ради них. Также я хочу поблагодарить моего брата, Тимура Шухратулловича Азимова, за разделение моих интересов и поддержку.

Наконец, я благодарен моим дорогим родителям, Шухратулло Сулханкуловичу и Елене Ивановне Азимовым, которые являются моей опорой на протяжении всей жизни и сделали меня тем, кто я есть сейчас.

Объем и структура работы. Диссертация состоит из введения, 6 глав и заключения. Полный объём диссертации составляет 135 страниц, включая 20 рисунков и 17 таблиц. Список литературы содержит 79 наименований.

Глава 1. Обзор

В данной главе введены основные термины и определения, используемые в работе, а также приводится формальная постановка задач поиска путей в графе с заданными КС-ограничениями и рассматриваются существующие алгоритмы для их решения. Кроме того, рассматриваются основные идеи использования методов линейной алгебры для решения задач анализа графов, а также существующие библиотеки линейной алгебры, которые могут быть использованы для получения соответствующих высокопроизводительных реализаций.

1.1 Основные понятия линейной алгебры

В данном разделе вводится ряд обозначений и определений линейной алгебры, используемых в работе.

Определение 1.1.1 (Алгебраическая структура)**.** *Алгебраическая структура* — это непустое множество $\mathcal{S}$ (носитель) с заданным набором операций и отношений (сигнатурой), которое удовлетворяет некоторой системе аксиом.

Далее приведём определения таких алгебраических структур, как полугруппа, моноид, группа и полукольцо.

Определение 1.1.2 (Полугруппа)**.** Множество $\mathcal{S}$ с заданной на нём бинарной операцией $\circ : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ называется *полугруппой* и обозначается $\langle \mathcal{S}, \circ \rangle$, если выполнена следующая аксиома.

- Ассоциативность: $\forall (a, b, c \in \mathcal{S}) \colon (a \circ b) \circ c = a \circ (b \circ c)$.

Определение 1.1.3 (Моноид)**.** *Моноидом* $\langle \mathcal{S}, \circ, \mathbb{0} \rangle$ называется такая полугруппа с некоторым выделенным элементом $\mathbb{0}$, что выполняется следующая аксиома.

- Наличие нейтрального элемента $\mathbb{0}$: $\forall a \in \mathcal{S} \colon (\mathbb{0} \circ a = a \circ \mathbb{0} = a)$.

Определение 1.1.4 (Группа)**.** Непустое множество $\mathcal{S}$ с заданной на нём бинарной операцией $\circ : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ и нейтральным элементом $\mathbb{0} \in \mathcal{S}$ называется *группой* $\langle \mathcal{S}, \circ, \mathbb{0} \rangle$, если оно является моноидом с дополнительным требованием наличия обратных элементов.

- Наличие обратного элемента: $\forall a \in \mathcal{S} \quad \exists a^{-1} \in \mathcal{S}\colon (a \circ a^{-1} = a^{-1} \circ a = \mathbb{0})$.

Определение 1.1.5 (Полукольцо)**.** Непустое множество $\mathcal{S}$ с двумя бинарными операциями $\oplus\colon \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ (часто называют сложением) и $\otimes\colon \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ (часто называют умножением), а также выделенными нейтральными элементами $\mathbb{0}, \mathbb{1} \in \mathcal{S}$ называется *полукольцом* $\langle \mathcal{S}, \oplus, \otimes, \mathbb{0}, \mathbb{1} \rangle$, если выполнены следующие условия.

1. $\langle \mathcal{S}, \oplus, \mathbb{0} \rangle$ — это коммутативный моноид, нейтральным элементом которого является $\mathbb{0}$, и при этом для любых $a, b, c \in \mathcal{S}$ имеем:
   - $(a \oplus b) \oplus c = a \oplus (b \oplus c)$,
   - $\mathbb{0} \oplus a = a \oplus \mathbb{0} = a$,
   - $a \oplus b = b \oplus a$.
2. $\langle \mathcal{S}, \otimes, \mathbb{1} \rangle$ — это моноид, нейтральным элементом которого является $\mathbb{1}$, и при этом для любых $a, b, c \in \mathcal{S}$ справедливо следующее:
   - $(a \otimes b) \otimes c = a \otimes (b \otimes c)$,
   - $\mathbb{1} \otimes a = a \otimes \mathbb{1} = a$.
3. Операция $\otimes$ является дистрибутивной слева и справа относительно операции $\oplus$, т.е. выполняется следующее:
   - $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$,
   - $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$.
4. Элемент $\mathbb{0}$ является *аннигилятором* для операции $\otimes$, т.е. имеем:
   - $\forall a \in \mathcal{S} : \mathbb{0} \otimes a = a \otimes \mathbb{0} = \mathbb{0}$.

Если операция $\otimes$ коммутативна, то говорят о коммутативном полукольце.

Кроме того, на носителях введённых алгебраических структур может быть задано отношение частичного порядка, которое определяется следующим образом.

Определение 1.1.6 (Отношение частичного порядка)**.** Бинарное отношение $\preceq$ на множестве $\mathcal{S}$ называется *отношением частичного порядка*, если для любых $a, b, c \in \mathcal{S}$ выполнены следующие условия.

- Рефлексивность: $a \preceq a$.
- Антисимметричность: если $a \preceq b$ и $b \preceq a$, то $a = b$.
- Транзитивность: если $a \preceq b$ и $b \preceq c$, то $a \preceq c$.

Далее введём понятия матрицы и вектора, а также определим некоторые операции над ними.

Определение 1.1.7 (Матрица)**.** Предположим, что имеется некоторая алгебраическая структура с носителем $\mathcal{S}$. Тогда *матрицей* будем называть прямоугольный массив размера $n \times m, n > 0, m > 0$, заполненный элементами из $\mathcal{S}$. Говорят, что n — это *высота* матрицы или количество строк в ней, а m — *ширина* матрицы или количество столбцов.

При доступе к элементам матрицы используются индексы. Индексация (нумерация) ведётся с левого верхнего угла, первым указывается строка элемента, вторым — его столбец. В работе будет использоваться индексация строк и столбцов, начинающаяся с нуля. Так, например, $M[0,0]$ — элемент матрицы, находящийся на пересечении верхней строки и левого столбца матрицы M.

Определение 1.1.8 (Вектор)**.** *Вектором* будем называть матрицу, хотя бы один из размеров которой равен единице. Если единице равна высота матрицы, то это *вектор-строка*, если же единице равна ширина матрицы, то это *вектор-столбец*.

Определение 1.1.9 (Скалярная операция)**.** Пусть $\langle \mathcal{S}, \circ \rangle$ является полугруппой, $M_{n\times m}$ — это матрица над этой полугруппой, $x \in \mathcal{S}$. Тогда результатом применения операции $scalar(M, x, \circ)$ является матрица $P_{n\times m}$ такая, что $P[i,j] = M[i,j] \circ x$, а $scalar(x, M, \circ) = P_{n\times m}$, где $P[i,j] = x \circ M[i,j]$.

Определение 1.1.10 (Поэлементные операции)**.** Пусть $\langle \mathcal{S}, \oplus, \otimes, \mathbb{0}, \mathbb{1} \rangle$ является полукольцом, а $M_{n\times m}$ и $N_{n\times m}$ — это две матрицы одинакового размера над этим полукольцом. Тогда результатом применения операции *поэлементного сложения* $\bigoplus$ является матрица $P_{n\times m} = M \bigoplus N$, которая определяется как $P[i,j] = M[i,j] \oplus N[i,j]$. А результатом применения операции *поэлементного умножения* $\bigotimes$ является матрица $P_{n\times m} = M \bigotimes N$, определённая как $P[i,j] = M[i,j] \otimes N[i,j]$.

Определение 1.1.11 (Умножение матриц)**.** Пусть $\langle \mathcal{S}, \oplus, \otimes, \mathbb{0}, \mathbb{1} \rangle$ является полукольцом, а $M_{n\times m}$ и $N_{m\times k}$ — это две матрицы над этим полукольцом. Тогда матрица $P_{n\times k} = M \cdot N$ определяется как $P[i,j] = \bigoplus_{0 \leqslant l < m} M[i,l] \otimes N[l,j]$.

Определение 1.1.12 (Произведение Кронекера)**.** Пусть $\langle \mathcal{S}, \circ \rangle$ — полугруппа, $M_{m\times n}$ и $N_{p\times q}$ — две матрицы над этой полугруппой. Тогда *произведение Кронекера* или тензорное произведение матриц M и N — это следующая блочная

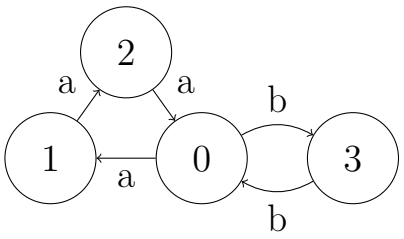


Рисунок 1.1 — Графическое представление графа $\mathcal{G}_1$

матрица C размера $mp \times nq$.

$$C = M \times N = \begin{pmatrix} scalar(M[0,0], N, \circ) & \cdots & scalar(M[0,n-1], N, \circ) \\ \vdots & \ddots & \vdots \\ scalar(M[m-1,0], N, \circ) & \cdots & scalar(M[m-1,n-1], N, \circ) \end{pmatrix}$$

1.2 Основные понятия теории графов

В данном разделе вводится ряд обозначений, а также представляется основная информация из теории графов.

Определение 1.2.1 (Граф)**.** *Граф* $\mathcal{G} = \langle V, E, L \rangle$, где V — конечное множество вершин, $E \subseteq V \times L \times V$ — конечное множество дуг, а L — конечное множество меток на дугах.

В дальнейшем речь будет идти о конечных ориентированных помеченных графах. Будем использовать термин *граф* подразумевая именно конечный ориентированный помеченный граф, если только не оговорено противное.

Также будем считать, что все вершины пронумерованы подряд, начиная с нуля. То есть, с каждой вершиной ассоциируется некоторое неотрицательное целое число из отрезка $[0, |V| - 1]$, где $|V|$ — количество вершин графа.

Пример 1.2.1 (Пример графа и его графического представления)**.** Пусть дан следующий граф:

$$\mathcal{G}_1 = \langle \{0,1,2,3\}, \{(0,a,1),(1,a,2),(2,a,0),(0,b,3),(3,b,0)\}, \{a,b\} \rangle.$$

Графическое представление графа $\mathcal{G}_1$ показано на рис. 1.1.

Определение 1.2.2 (Дуга)**.** *Дуга* ориентированного помеченного графа $\mathcal{G} = \langle V, E, L\rangle$ — это упорядоченная тройка $e = (v_i, l, v_j) \in V \times L \times V$.

Пример 1.2.2 (Пример дуг графа)**.** $(0, a, 1)$ и $(3, b, 0)$ — это дуги графа $\mathcal{G}_1$. При этом, $(3, b, 0)$ и $(0, b, 3)$ — это разные дуги, что видно из рис. 1.1.

Определение 1.2.3 (Кратные дуги)**.** Две различные дуги $e_1 = (u_1, l_1, v_1)$ и $e_2 = (u_2, l_2, v_2)$ ориентированного помеченного графа $\mathcal{G} = \langle V, E, L\rangle$ называются *кратными*, если $u_1 = u_2$ и $v_1 = v_2$.

Определение 1.2.4 (Путь)**.** *Путём* π в графе $\mathcal{G}$ будем называть последовательность дуг такую, что для любых двух последовательных дуг $e_1 = (u_1, l_1, v_1)$ и $e_2 = (u_2, l_2, v_2)$ в этой последовательности, конечная вершина первой дуги является начальной вершиной второй, то есть $v_1 = u_2$. Будем обозначать путь из вершины v_0 в вершину v_n следующим образом:

$$v_0 \pi v_n = e_0, e_1, \ldots, e_{n-1} = (v_0, l_0, v_1), (v_1, l_1, v_2), \ldots, (v_{n-1}, l_n, v_n).$$

Определение 1.2.5 (Цикл)**.** *Циклом* называется путь π в графе $\mathcal{G}$ такой, что его начальная и конечная вершины совпадают, а все дуги — различны.

Будем считать, что для множества меток L всегда определена операция конкатенации $(\cdot) : L^* \times L^* \to L^*$. При записи выражений символ точки (обозначение операции конкатенации) часто будем опускать: $l_1 \cdot l_2 = l_1 l_2$.

Определение 1.2.6 (Слово, образованное путём)**.** *Слово, образованное путём* $\pi = (v_0, l_0, v_1), (v_1, l_1, v_2), \ldots, (v_{n-1}, l_n, v_n)$, будем обозначать как слово $\lambda(\pi) = l_0 l_1 \ldots l_n$, которое является конкатенацией меток всех дуг данного пути слева направо.

Пример 1.2.3 (Пример путей графа)**.** $(0, a, 1), (1, a, 2) = 0\pi_1 2$ — путь из вершины 0 в вершину 2 в графе $\mathcal{G}_1$. При этом, $(0, a, 1), (1, a, 2), (2, b, 3), (3, b, 2) = 0\pi_2 2$ — это тоже путь из вершины 0 в вершину 2 в графе $\mathcal{G}_1$, но он не равен $0\pi_1 2$. Причем, $\lambda(\pi_1) = aa$ и $\lambda(\pi_2) = aabb$.

Также определим понятие отрицательного цикла для взвешенных графов, в которых с каждым ребром ассоциируется вещественное число и которые являются частным случаем помеченных графов.

Определение 1.2.7 (Отрицательный цикл)**.** *Отрицательным циклом* во взвешенном графе $\mathcal{G} = \langle V, E, \mathbb{R} \rangle$ называется цикл с отрицательной суммой весов на его дугах.

Кроме того, нам потребуется отношение, отражающее факт существования пути между двумя вершинами.

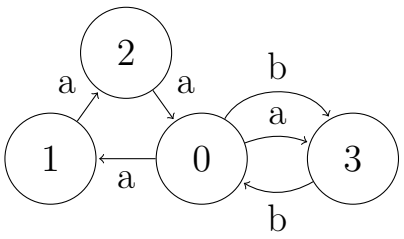
Определение 1.2.8 (Отношение достижимости)**.** *Отношение достижимости* в графе: $(v_i, v_j) \in P \iff \exists v_i \pi v_j$.

Также определим понятие матрицы смежности, что позволит использовать один из распространённых способов для представления графов в виде матриц.

Определение 1.2.9 (Матрица смежности)**.** *Матрица смежности* графа $\mathcal{G} = \langle V, E, L \rangle$ — это квадратная матрица M размера $n \times n$, где $|V| = n$ и ячейки которой содержат множества меток на дугах этого графа. При этом $l \in M[i, j] \iff \exists e = (i, l, j) \in E$.

Заметим, что наше определение матрицы смежности отличается от классического, в котором матрица отражает лишь факт наличия хотя бы одной дуги и, соответственно, является булевой.

Пример 1.2.4 (Пример графа и его матрицы смежности)**.** Пример помеченного графа представлен ниже:



Матрица смежности для этого графа будет выглядеть так:

$$M = \begin{pmatrix} \varnothing & \{a\} & \varnothing & \{a,b\} \\ \varnothing & \varnothing & \{a\} & \varnothing \\ \{a\} & \varnothing & \varnothing & \varnothing \\ \{b\} & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

Такие матрицы смежности могут быть представлены в виде множества булевых матриц. Для краткости значение булевой переменной $true$ будем обозначать единицей, а $false$ — нулём.

Определение 1.2.10 (Булева декомпозиция матрицы смежности)**.** *Булева декомпозиция матрицы смежности* M графа $\mathcal{G} = \langle V, E, L \rangle$ — это множество булевых матриц $\mathcal{M} = \{M^l \mid l \in L, M^l[i,j] = 1 \iff l \in M[i,j]\}$.

Пример 1.2.5 (Пример булевой декомпозиции матрицы смежности M из примера 1.2.4)**.** Булева декомпозиция $\mathcal{M}$ состоит из матриц M^a и M^b.

$$M^a = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M^b = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Определение 1.2.11 (Произведение Кронекера двух графов)**.** Пусть даны два помеченных графа $\mathcal{G}_1 = \langle V_1, E_1, L_1 \rangle$ и $\mathcal{G}_2 = \langle V_2, E_2, L_2 \rangle$. Тогда *произведение Кронекера двух графов* $\mathcal{G}_1$ и $\mathcal{G}_2$ — это граф $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2 = \langle V, E, L \rangle$, где

- $V = V_1 \times V_2$,
- $E = \{((u,v), l, (p,q)) \mid (u,l,p) \in E_1 \wedge (v,l,q) \in E_2\}$,
- $L = L_1 \cap L_2$.

Из этого определения, а также из определения 1.1.12 следует, что результатом произведения Кронекера матриц смежности двух графов $\mathcal{G}_1$ и $\mathcal{G}_2$, определённого для полугруппы $\langle 2^{L_1} \cup 2^{L_2}, \cap \rangle$, является матрица смежности произведения Кронекера этих графов $\mathcal{G}_1 \times \mathcal{G}_2$. Здесь 2^L — множество всех подмножеств множества L.

1.3 Основные понятия теории формальных языков

В данном разделе вводится ряд обозначений, а также представляется основная информация из теории формальных языков.

Определение 1.3.1 (Алфавит)**.** *Алфавит* Σ — это конечное множество некоторых элементов. Элементы этого множества будем называть *символами*.

Как и в случае с множеством меток на дугах графа будем считать, что для алфавита Σ всегда определена операция конкатенации $(\cdot) : \Sigma^* \times \Sigma^* \to \Sigma^*$.

Определение 1.3.2 (Слово)**.** *Слово* над алфавитом Σ — это конечная конкатенация символов алфавита Σ: $w = a_0 \cdot a_1 \cdot \ldots \cdot a_m$, где w — слово для любых $a_i \in \Sigma$.

Определение 1.3.3 (Длина слова)**.** Пусть $w = a_0 \cdot a_1 \cdot \ldots \cdot a_m$ — слово над алфавитом Σ. Будем называть $m+1$ *длиной слова* и обозначать как $|w|$. Кроме того, для обозначения пустого слова будем использовать символ ε, где $|\varepsilon| = 0$.

Определение 1.3.4 (Язык)**.** *Язык* над алфавитом Σ — это множество слов над алфавитом Σ.

В качестве примера простейших языков можно привести язык чисел в двоичной системе счисления $\{0, 1, 10, 11, 100 \ldots\}$, а также язык всех правильных скобочных последовательностей $\{(), (()), ()(), (())(), \ldots\}$.

Любой язык над алфавитом Σ является подмножеством Σ^* — множества всех слов над алфавитом Σ. Заметим, что язык может является бесконечным множеством.

Определение 1.3.5 (Формальная грамматика)**.** *Формальная грамматика* G — это четвёрка $\langle \Sigma, N, P, S \rangle$:

- Σ обозначает конечный алфавит терминальных символов или терминалов,
- N — алфавит нетерминальных символов или нетерминалов, $\Sigma \cap N = \varnothing$,
- P — конечное подмножество множества

$$(\Sigma^* \cdot N \cdot (\Sigma \cup N)^*) \times ((\Sigma \cup N)^+ \cup \{\varepsilon\}),$$

- S — стартовый символ грамматики, $S \in N$.

Элемент $(a, b) \in P$ называется правилом вывода и записывается следующим образом: $a \to b$. При этом a называется левой частью правила, а b — правой частью. Левая часть любого правила из P обязана содержать хотя бы один нетерминал.

Определение 1.3.6 (Вывод цепочки w в грамматике G)**.** Цепочка $w_2 \in (\Sigma \cup N)^*$ *непосредственно выводима* из цепочки $w_1 \in (\Sigma \cup N)^+$ в грамматике $G = \langle \Sigma, N, P, S \rangle$ (обозначается $w_1 \to_G w_2$), если $w_1 = x_1 \cdot y \cdot x_2, w_2 = x_1 \cdot z \cdot x_2$, где $x_1, x_2, z \in (\Sigma \cup N)^*, y \in (\Sigma \cup N)^+$ и правило вывода $y \to z$ содержится в P. Индекс G в обозначении $\to_G$ обычно опускают, если G понятна из контекста.

Цепочка $w_2 \in (\Sigma \cup N)^*$ *выводима* из цепочки $w_1 \in (\Sigma \cup N)^+$ в грамматике G (обозначается $w_1 \Rightarrow_G w_2$), если существуют цепочки $z_0, z_1, \ldots, z_n (n \geqslant 0)$ такие, что $w_1 = z_0 \to z_1 \to \ldots \to z_n = w_2$. При этом последовательность $z_0, z_1, \ldots, z_n$ называется выводом длины n.

Определение 1.3.7 (Язык, порождаемый грамматикой)**.** Язык, *порождаемый* грамматикой $G = \langle \Sigma, N, P, S \rangle$ — это множество $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \Rightarrow w\}$.

Важными классами языков, порождаемых формальными грамматиками, являются регулярные и контекстно-свободные языки [35]. Одним из распространённых способов описания регулярных языков (порождаемых регулярными грамматиками) является их задание с использованием регулярных выражений.

Определение 1.3.8 (Регулярное выражение)**.** Цепочка символов R называется *регулярным выражением* над алфавитом Σ, если её структура соответствует одному из следующих пунктов:

- a, где $a \in \Sigma$;
- ε, где ε — пустая строка;
- $\varnothing$ (соответствует пустому языку);
- $R_1 \mid R_2$ (дизъюнкция), где R_1 и R_2 являются регулярными выражениями;
- $R_1 \cdot R_2$ (конкатенация), где R_1 и R_2 являются регулярными выражениями;
- $(R_1)^*$ (звезда Клини), где R_1 — это регулярное выражение.

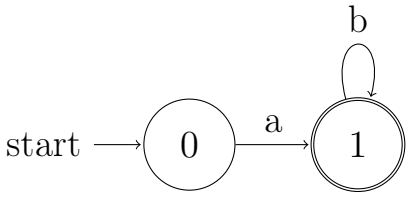
Например, регулярное выражение $R = a \cdot (b)^*$ над алфавитом $\Sigma = \{a, b\}$ описывает язык, состоящий из слов, которые начинаются с символа a с последующим некоторым (ноль или больше) количеством символов b.

Определение 1.3.9 (Конечный автомат)**.** *Детерминированный конечный автомат без ε-переходов* T — пятёрка $\langle \Sigma, Q, Q_s, Q_f, \delta \rangle$:

- Σ — алфавит входных символов,
- Q — конечное множество состояний,
- $q_s \in Q$ — начальное состояние,
- $Q_f \subseteq Q$ — множество конечных состояний,
- $\delta : Q \times \Sigma \rightharpoonup Q$ — частично определённая функция переходов.

Известно, что любое регулярное выражение может быть выражено с помощью детерминированного конечного автомата без ε-переходов [36]. Кроме того, конечный автомат $T = \langle \Sigma, Q, q_s, Q_f, \delta \rangle$ естественным образом может быть представлен в виде помеченного графа $\mathcal{G} = \langle V, E, L \rangle$, где $V = Q$, $L = \Sigma$, $E = \{(q_i, l, q_j) \mid \delta(q_i, l) = q_j\}$ и некоторые вершины помечены как начальное или конечные состояния. Таким образом, матрица смежности такого графового представления и, соответственно, её булева декомпозиция содержат в себе информацию о функции переходов δ.

Пример 1.3.1 (Пример представления в виде графа конечного автомата для регулярного выражения $a \cdot (b)^*$)**.** Вершину графа, соответствующую стартовому состоянию, будем помечать с помощью слова *start*, а вершины, соответствующие конечным состояниям, будем помечать двойным кругом. В данном примере стартовому состоянию соответствует вершина 0, а конечному — вершина 1.



Определение 1.3.10 (Пересечение конечных автоматов)**.** Пусть даны два конечных автомата $T_1 = \langle \Sigma, Q^1, q_s^1, Q_f^1, \delta^1 \rangle$ и $T_2 = \langle \Sigma, Q^2, q_s^2, Q_f^2, \delta^2 \rangle$. Тогда *пересечением* этих двух автоматов является новый конечный автомат $T = \langle \Sigma, Q, q_s, Q_f, \delta \rangle$ такой, что:

- $Q = Q^1 \times Q^2$;
- $q_s = \langle q_s^1, q_s^2 \rangle$;

- $Q_f = Q_f^1 \times Q_f^2$;
- $\delta : Q \times \Sigma \rightharpoonup Q$;
- $\delta(\langle q_1, q_2 \rangle, s) = \langle q_1', q_2' \rangle$, если $\delta(q_1, s) = q_1'$ и $\delta(q_2, s) = q_2'$.

Согласно [36] конечный автомат T, являющийся пересечением конечных автоматов T_1 и T_2 распознаёт язык $L_1 \cap L_2$, где L_1 — язык, распознаваемый автоматом T_1, а L_2 — язык, распознаваемый автоматом T_2. Кроме того, построение пересечения двух конечных автоматов может быть проведено с использованием булевой декомпозиции матриц смежности графовых представлений этих автоматов.

Определение 1.3.11 (Произведение Кронекера булевых декомпозиций матриц смежности)**.** Пусть дана полугруппа $\langle \{0, 1\}, \wedge \rangle$ и пусть $\mathcal{M}_1$ и $\mathcal{M}_2$ — булевы декомпозиции матриц смежности графовых представлений конечных автоматов $T_1 = \langle \Sigma, Q^1, q_s^1, Q_f^1, \delta^1 \rangle$ и $T_2 = \langle \Sigma, Q^2, q_s^2, Q_f^2, \delta^2 \rangle$. Тогда в соответствии с заданной полугруппой *произведение Кронекера булевых декомпозиций матриц смежности двух конечных автоматов* $\mathcal{M}_1 \times \mathcal{M}_2 = \{M_1^a \times M_2^a \mid a \in \Sigma\}$.

Согласно [37] справедлива следующая теорема.

Теорема 1.3.1 (Пересечение конечных автоматов и произведение Кронекера)**.** Пусть T — пересечение конечных автоматов T_1 и T_2. Пусть $\mathcal{M}$, $\mathcal{M}_1$ и $\mathcal{M}_2$ — булевы декомпозиции матриц смежности графовых представлений конечных автоматов T, T_1 и T_2. Тогда $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2$.

Таким образом, функция переходов пересечения двух конечных автоматов может быть вычислена с использованием ряда произведений Кронекера над булевыми матрицами.

Другим важным классом формальных языков, обобщающим класс регулярных языков, являются контекстно-свободные (КС) языки, порождаемые грамматиками следующего вида.

Определение 1.3.12 (Контекстно-свободная грамматика)**.** *Контекстно-свободная (КС) грамматика* G — это четвёрка $\langle \Sigma, N, P, S \rangle$, где

- Σ обозначает конечный алфавит терминальных символов,
- N — алфавит нетерминальных символов, $\Sigma \cap N = \varnothing$,
- P — конечное подмножество множества $N \times ((\Sigma \cup N)^+ \cup \{\varepsilon\})$,
- S — стартовый символ грамматики, $S \in N$.

Для представления вывода цепочки в КС-грамматиках обычно используют деревья вывода.

Определение 1.3.13 (Дерево вывода цепочки в КС-грамматике)**.** *Деревом вывода цепочки* $w \in \Sigma^*$ в КС-грамматике $G = \langle \Sigma, N, P, S \rangle$ называется упорядоченное корневое дерево со следующими свойствами.

- Корень помечен S.
- Если его внутренний узел помечен $A \in N$ и $X_1, \ldots, X_k \in \Sigma \cup N$ — перечисленные слева направо пометки всех сыновей этого узла, то правило $A \to X_1 \ldots X_k \in P$.
- Если его внутренний узел помечен $A \in N$ и ε — пометка единственного сына этого внутреннего узла, то правило $A \to \varepsilon \in P$.
- $w = a_1 \ldots a_m$, где $a_1, \ldots, a_m \in \Sigma \cup \{\varepsilon\}$ перечисленные слева направо пометки всех листьев этого дерева.

Часто, для алгоритмов, использующих КС-грамматики, важно, чтобы она находилась в некоторой нормальной форме. Примером является нормальная форма следующего вида, которая будет в дальнейшем использоваться в данной работе.

Определение 1.3.14 (Ослабленная нормальная форма Хомского КС-грамматики)**.** Будем говорить, что КС-грамматика $G = \langle \Sigma, N, P, S \rangle$ находится в *ослабленной нормальной форме Хомского*, если множество P содержит только правила следующих двух видов:

- $A \to a$, где $A \in N$ и $a \in (\Sigma \cup \{\varepsilon\})$;
- $A \to BC$, где $A, B, C \in N$.

Стоит отметить, что описанная нормальная форма КС-грамматики отличается от классической нормальной формы Хомского [38], в которой не допускаются правила вида $A \to \varepsilon$ для всех нетерминалов A, кроме стартового S. Однако это ограничение не является существенным для использования в данной работе, поэтому будет использоваться нормальная форма более общего вида.

В то время, как регулярное выражение может быть записано в виде конечного автомата, любая КС-грамматика может быть выражена с помощью рекурсивного автомата [39].

Определение 1.3.15 (Рекурсивный автомат)**.** *Рекурсивный автомат* R — четвёрка $\langle \Sigma, B, m, \{C_i\}_{i \in B} \rangle$:

- Σ — конечный алфавит входных символов,
- B — конечное множество меток автоматов,
- $m \in B$ — метка стартового автомата,
- Множество автоматов $C_i = \langle \Sigma \cup B, Q_i, q_s^i, Q_f^i, \delta_i \rangle$:
  - $\Sigma \cup B$ — множество допустимых символов переходов автомата, где $\Sigma \cap B = \varnothing$;
  - Q_i — конечное множество состояний автомата, где $Q_i \cap Q_j = \varnothing$, $\forall i \neq j$;
  - q_s^i — начальное состояние для автомата C_i;
  - $Q_f^i \subseteq Q_i$ — множество конечных состояний,
  - $\delta_i : Q_i \times (\Sigma \cup B) \to Q_i$ — функция переходов.

Рекурсивные автоматы в свою очередь могут быть представлены в виде графов или матриц смежности этих графов [39].

Пример 1.3.2 (Пример представления в виде графа рекурсивного автомата для КС-грамматики, порождающей КС-язык $\mathcal{L} = \{a^n b^n \mid n \geqslant 1\}$)**.** Пусть дана КС-грамматика G, правила вывода которой имеют следующий вид.

$$\begin{aligned} 0: &\quad S \;\to\; a\ S\ b \\ 1: &\quad S \;\to\; a\ b \end{aligned}$$

Тогда рекурсивный автомат $R = \langle \{a, b\}, \{S\}, S, \{C_S\} \rangle$ для КС-грамматики G представлен на рис. 1.2. Метки автоматов будем указывать в левом верхнем углу. В данном примере рекурсивный автомат содержит только один конечный автомат C_S с меткой S, соответствующей стартовому нетерминалу грамматики.

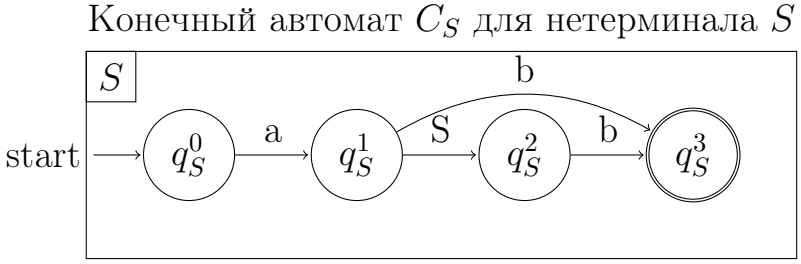


Рисунок 1.2 — Рекурсивный автомат R для КС-грамматики G

Таким образом, любой КС-язык может быть описан с использованием таких объектов линейной алгебры, как матрицы.

1.4 Постановка задач поиска путей в графе с КС-ограничениями

Для исследования применимости методов линейной алгебры к задачам поиска путей в графе с заданными КС-ограничениями необходимо сначала ввести формальную постановку этих задач. В данном разделе вводится формальная постановка для трёх основных типов таких задач.

В этих задачах на вход подаётся помеченный граф $\mathcal{G} = \langle V, E, L \rangle$ и КС-язык $\mathcal{L}$. Выходом является информация о путях π входного графа $\mathcal{G}$ таких, что $\lambda(\pi) \in \mathcal{L}$. Вид запрашиваемой информации зависит от типа решаемой задачи. Выделяют три типичных задачи поиска путей в графе:

- задача достижимости,
- поиск одного пути,
- поиск всех путей.

Ниже представлена формальная постановка для каждой из этих задач.

Задача достижимости. Пусть даны помеченный граф $\mathcal{G} = \langle V, E, L \rangle$ и КС-язык $\mathcal{L}$. Необходимо найти все пары вершин $i, j \in V$, между которыми существует хотя бы один путь π такой, что $\lambda(\pi) \in \mathcal{L}$. То есть необходимо построить следующее множество:

$$\{(i, j) \mid i, j \in V, \exists i\pi j, \lambda(\pi) \in \mathcal{L}\}.$$

Поиск одного пути. Пусть даны помеченный граф $\mathcal{G} = \langle V, E, L \rangle$ и КС-язык $\mathcal{L}$. Необходимо для всех пар вершин $i, j \in V$ предоставить один путь π такой, что $\lambda(\pi) \in \mathcal{L}$, если такой путь существует.

Поиск всех путей. Пусть даны помеченный граф $\mathcal{G} = \langle V, E, L \rangle$ и КС-язык $\mathcal{L}$. Необходимо для всех пар вершин $i, j \in V$ предоставить любое конечное заданное количество путей π таких, что $\lambda(\pi) \in \mathcal{L}$, если существует такое количество путей.

Проиллюстрируем постановки данных задач на небольшом графе с использованием классического КС-языка $\mathcal{L} = \{a^n b^n \mid n \geqslant 1\}$, который не может быть описан с помощью регулярного выражения и который используется для нахождения вершин в графе, находящихся на одном уровне иерархии [40].

Пусть имеется графовая база данных или любой другой объект, который может быть представлен в виде графа. Тогда КС-язык $\mathcal{L}$ может быть использован для выявления схожих вершин в этом графе. Для графовых баз

данных такая задача нацелена на поиск всех вершин, находящихся на одинаковом уровне иерархии. Например, рассмотрим небольшой граф с двумя циклами, представленный на рис. 1.1. Один из циклов имеет три дуги с метками a, другой — две дуги с метками b. Оба цикла имеют общую вершину с номером 0.

Пусть результатом решения задачи достижимости с заданными КС-ограничениями является множество пар вершин R. Тогда, например, пара вершин $(0,0) \in R$, так как существует путь из вершины 0 в вершину 0, образующий строку $w = aaaaaabbbbbb = a^6b^6 \in \mathcal{L}$.

Результат решения задачи поиска одного пути в графе с заданными КС-ограничениями также содержит по одному примеру пути искомого вида между каждой парой вершин $(i,j) \in R$. Например, если мы хотим предоставить доказательство существования пути искомого вида между вершинами 0 и 0, то в результат может быть добавлен путь $\pi = (0,a,1),(1,a,2),(2,a,0),(0,a,1),(1,a,2),(2,a,0),(0,b,3),(3,b,0),(0,b,3),(3,b,0),(0,b,3),(3,b,0)$ для пары $(0,0) \in R$, где $\lambda(\pi) = a^6b^6 \in \mathcal{L}$.

Результат решения задачи поиска всех путей в графе с заданными КС-ограничениями должен содержать все пути искомого вида между каждой парой вершин $(i,j) \in R$. Например, для пары вершин $(0,0) \in R$ должны быть представлены пути $0\pi_k0$, где $\lambda(\pi_k) = a^{6k}b^{6k}$ для всех $k \geqslant 1$. Множество всех путей может быть бесконечным, поэтому необходимо либо представить данное множество в виде некоторой конечной структуры, либо иметь алгоритм генерации k путей для каждой пары вершин и для любого $k \geqslant 1$.

1.5 Обзор алгоритмов поиска путей в графе с КС-ограничениями

В данном разделе рассмотрены основные существующие алгоритмы поиска путей в графе с КС-ограничениями. Все рассматриваемые алгоритмы требуют выразить заданные КС-ограничения в виде КС-грамматики, поэтому будем считать, что на вход алгоритмы получают помеченный граф $\mathcal{G} = \langle V, E, L\rangle$ и КС-грамматику $G = \langle \Sigma, N, P, S\rangle$.

Алгоритмы для произвольных графов с произвольными КС-ограничениями. Существует ряд алгоритмов для задачи достижимости с заданными КС-ограничениями, основанных на различных алгоритмах синтак-

сического анализа. Так, например, Джелле Хеллингс в работе [9] предложил алгоритм с временной сложностью $O(|N||E| + (|N||V|)^3)$, основанный на алгоритме синтаксического анализа CYK. Кроме того, в работе [5] была получена реализация этого алгоритма для анализа RDF данных. Также существуют работы [24] и [25], в которых для решения задачи достижимости с заданными КС-ограничениями использовались LL и LR алгоритмы синтаксического анализа [35]. Для алгоритма в работе [24], основанного на LL алгоритме, также известна временная сложность в худшем случае: $O(|V|^3|P|)$. С практической точки зрения, в работе [14] было проведено исследование, в котором сравнивались реализации основных существующих алгоритмов для задачи достижимости с заданными КС-ограничениями. Это исследование показало, что существующие решения недостаточно производительны для использования на практике.

Кроме того, Джелле Хеллингс в работе [10] предложил алгоритм поиска одного пути в графе с заданными КС-ограничениями. Для этого строится так называемая «аннотированная» грамматика, которая является конечной структурой, содержащей в себе информацию обо всех искомых путях графа. Далее используются результаты из теории формальных языков для нахождения кратчайшей строки, порождаемой КС-грамматикой. Для каждой пары вершин такие строки используются для нахождения кратчайшего пути между ними, удовлетворяющего заданным КС-ограничениям. Нахождение таких путей решает поставленную задачу поиска одного пути. Временная сложность предложенного алгоритма — $O(|N||V|^2(|N||V|^2 log(|N||V|^2) + |P|(|V|^3 + |E|)) + L)$, где L — сумма длин найденных кратчайших путей в графе.

Также существует ряд алгоритмов [26; 41; 42] поиска всех путей в графе с заданными КС-ограничениями, основанных на алгоритмах синтаксического анализа GLL [43] и GLR [44]. В предложенных алгоритмах для хранения информации об искомых путях используется конечное представления леса разбора SPPF [45]. В дальнейшем эта структура используется для генерации любого количества путей, удовлетворяющих заданным КС-ограничениям. Кроме того, для алгоритма, предложенного в работе [26] известна временная сложность построения такого леса разбора SPPF в худшем случае: $O(|V|^3 \max_{v \in V} deg^+(v))$, где $deg^+(v)$ — количество исходящих дуг из вершины v. Также существует работа [46], в которой Хеллингс аналогично задаче поиска одного пути с заданными КС-ограничениями строит «аннотированную» грамматику. Однако построен-

ная грамматика уже содержит всю необходимую информацию обо всех путях в графе, удовлетворяющих заданным КС-ограничениям. Затем эта грамматика может быть использована для генерации искомых путей в графе. Временная сложность построения такой грамматики — $O(|N||E|+(|N||V|)^3)$, однако сложность генерации искомых строк пока что мало исследована.

Алгоритмы для частных случаев задачи поиска путей в графе. Кроме того, в ряде работ [7; 47; 48] было показано, как можно проводить статический анализ программ, решая задачу достижимости в графе с заданными КС-ограничениями. Затем эта задача была использована для формулировки различных проблем статического анализа программ таких, как анализ указателей и поиск псевдонимов [2; 49—53], анализ зависимостей в данных [49], анализ вывода типов [54], анализ потока данных с использованием системы типов [1] и т.д. В этих алгоритмах графы, построенные из программ, анализируются с использованием определенного КС-языка в качестве ограничений на пути. Примером такого языка является язык Дика [7] (язык правильных скобочных последовательностей). Поэтому предложенные алгоритмы предназначены для решения частного случая задачи достижимости с заданными КС-ограничениями.

Другим примером алгоритма для решения частного случая задачи достижимости с заданными КС-ограничениями является алгоритм, предложенный Филипом Брэдфордом в работе [8] и использующий операции над матрицами. Этот алгоритм имеет временную сложность в худшем случае $O(|V|^{\omega}log^3|V|)$, где ω — наилучший показатель степени для оценки временной сложности умножения двух матриц размера $n \times n$. Стоит отметить, что алгоритм был также сформулирован с использованием методов линейной алгебры, однако он предназначен только для частного случая КС-ограничений, выраженных с помощью языка Дика с одним типом скобок.

Кроме того, в работе [55] был предложен алгоритм, решающий задачу достижимости в графе с заданными КС-ограничениями, который был использован для извлечения из графа информации, детально описывающей получение того или иного результата анализа графа. Для решения такой задачи использовался определённый КС-язык, который не может быть описан регулярными выражениями и является языком некоторых палиндромов [36]. Для такого КС-языка и для задачи достижимости в графе с заданными КС-ограничениями

с фиксированным набором конечных вершин V_{dst} был предложен алгоритм с линейной сложностью, если рассматривать $|V_{dst}|$ как константу.

1.6 Использование методов линейной алгебры для анализа графов

В данном разделе представлены основные идеи использования методов линейной алгебры для решения задач анализа графов, а также подход к решению задач поиска путей в графах, основанный на этих идеях.

1.6.1 Основные идеи

В последнее время многие задачи анализа графов решают с использованием методов линейной алгебры, что, в свою очередь, позволяет использовать имеющиеся средства распараллеливания, например, для умножения матриц, добиваясь эффективной реализации данных алгоритмов на GPU. При этом, как правило, берётся готовый алгоритм, решающий нужную задачу, и далее этот алгоритм переводится на язык линейной алгебры. Для выполнения этого «перевода» можно выделить следующие шаги.

1. Представление основных сущностей алгоритма — графа, множества рассматриваемых на текущей итерации вершин и пр. — в виде объектов линейной алгебры, т.е. матриц и векторов.
2. Выбор процедуры обхода графа в виде набора операций над объектами линейной алгебры.
3. Реализация семантической части алгоритма в виде модификации выбранных операций над объектами линейной алгебры. При этом используются дополнительные алгебраические структуры (полугруппы, моноиды, полукольца и пр), над которыми строятся объекты линейной алгебры.

Ключевым является выбор объектов для представления информации о графах. Например, сам граф можно описать матрицей смежности, а стро-

ка/столбец этой матрицы описывает исходящие/входящие дуги вершины, задаваемой номером строки/столбца.

После выбора подходящих объектов важно определить процедуру обхода графа. По сути, этот процесс заключается в рассмотрении всех необходимых путей, и различные методы обхода графов предлагают различный порядок рассмотрения вершин, дуг и путей графа, иными словами имеют различные шаблоны доступа к данным. При удачном выборе процедуры обхода графа и, соответственно, шаблона доступа к данным, в линейной алгебре могут найтись операции над выбранными объектами, имеющие схожий или совпадающий шаблон. Тогда сам процесс обхода графа и рассмотрения различных путей может быть проведен с помощью этих операций.

Однако, если мы решаем не просто задачу о достижимости одной вершины из другой, то кроме обхода графа необходимо ещё выполнить определённые действия при обходе. Например, вычисление некоторой информации о рассматриваемых путях или проверку путей на соответствие некоторым ограничениям в зависимости от поставленной задачи (поиск кратчайших путей, простых путей, поиск путей с заданными КС-ограничениями и т.д.). Эти действия могут быть реализованы дополнительно, однако некоторые операции линейной алгебры могут быть модифицированы с целью выполнения всех необходимых для решения поставленной задачи действий. Поэтому целесообразно исследовать возможность таких модификации для операций, которые соответствуют выбранной процедуре обхода графа.

Например, на листинге 1 представлен псевдокод алгоритма Беллмана-Форда [56; 57] поиска кратчайших расстояний во взвешенном графе без кратных дуг с множеством вершин V, множеством дуг E, функции весов W и выделенной стартовой вершиной $s \in V$. При условии отсутствия отрицательных циклов во входном графе данный алгоритм возвращает кратчайшие расстояния от стартовой вершины до всех остальных.

Согласно [13] этот алгоритм может быть переведён на язык линейной алгебры следующим образом. Граф был представлен в виде матрицы смежности, а для хранения расстояний от стартовой вершины s до всех остальных вершин использовать вектор d. Заметим, что из-за отсутствия кратных дуг в графе элементами матрицы смежности являются множества, состоящие не более чем из одного вещественного числа. Поэтому для удобства преобразуем матрицу смежности и будем рассматривать матрицу A с элементами — вещественными

числами или специальными элементами ∞, обозначающими отсутствие дуги между двумя вершинами. Процедура обхода графа задаётся с помощью умножения вектора на матрицу, как это представлено в листинге 2. В представленном обходе графа весь процесс поиска путей сводится к вычислению ряда таких умножений $d \cdot A$ в строке 5. Однако для решения задачи поиска кратчайших путей необходимо задать дополнительные действия в процессе обхода графа. Для этого операция умножения вектора на матрицу переопределяется с использованием такой алгебраической структуры, как полукольцо $\langle \mathbb{R} \cup \{\infty\}, min, +, \infty, 0 \rangle$, которое позволяет вместо привычной операции сложения элементов матриц и векторов использовать операцию взятия минимума, а вместо операции умножения этих элементов использовать операцию сложения вещественных чисел. При использовании этого полукольца операция min позволяет из нескольких альтернативных путей между двумя вершинами выбирать тот, который имеет меньшую длину, а операция + позволяет вычислять длины рассматриваемых путей, как суммы длин их подпутей.

Листинг 1 Алгоритм Беллмана-Форда поиска кратчайших расстояний во взвешенном графе

```
1:  function BELLMAN-FORD(V — множество вершин, E — множество дуг,
    W — функция весов, s — стартовая вершина)
2:      d ← список кратчайших расстояний от вершины s до любой другой
3:      for all v ∈ V do
4:          d(v) ← ∞
5:      d(s) ← 0                          ▷ Расстояние до стартовой вершины
6:      for k = 1 to N − 1 do
7:          if d(v) > d(u) + W(u, v) then
8:              d(v) ← d(u) + W(u, v)
9:      for all (u, v) ∈ E do
10:         if d(v) > d(u) + W(u, v) then
11:             return "В графе существует отрицательный цикл."
12:     return d
```

Таким образом, удачный выбор объектов линейной алгебры, процедуры обхода графа и дополнительных действий в процессе этого обхода позволяют строить алгоритмы для решения задач анализа графов с помощью операций линейной алгебры.

Листинг 2 Процедура обхода графа для алгоритма Беллмана-Форда с использованием умножения вектора на матрицу

```
1: function ALGEBRAIC-BELLMAN-FORD(A — матрица смежности графа, s — стартовая вершина)
2:     d ← вектор кратчайших расстояний от вершины s до любой другой со всеми элементами равными ∞
3:     d(s) ← 0                                  ▷ Расстояние до стартовой вершины
4:     for k = 1 to N − 1 do
5:         d ← d · A                             ▷ Умножение вектора на матрицу
6:     if d ≠ d · A then
7:         return "В графе существует отрицательный цикл."
8:     return d
```

1.6.2 Подход Algebraic Path Problem

Представленные в предыдущем разделе идеи решения задач анализа ориентированных помеченных графов легли в основу подхода, называемого *Algebraic Path Problem* [58]. Суть этого подхода заключается в построении полукольца для решения задачи анализа графа, и это полукольцо позволяет использовать алгебраические операции для вычисления необходимых свойств путей в графе. Более формально, пусть дан граф $\mathcal{G} = \langle V, E, L \rangle$. Для этого графа предлагается построить такое полукольцо $\langle \mathcal{S}, \oplus, \otimes, \mathbb{0}, \mathbb{1} \rangle$, в котором $L \subseteq \mathcal{S}$ и которое позволяет свести изначальную задачу анализа графа к вычислению для всех $i, j \in V$ значений $d_{i,j} = \bigoplus\{\lambda_{algebraic}(\pi) \mid \pi$ — путь из вершины i в вершину $j\}$. При этом для пути $\pi = (v_0, l_0, v_1), (v_1, l_1, v_2), \ldots, (v_{n-1}, l_n, v_n)$ выполнено $\lambda_{algebraic}(\pi) = (\ldots(l_0 \otimes l_1) \otimes l_2)\ldots) \otimes l_n$.

Такие значения могут быть вычислены с использованием метода Гаусса [59—61], на основе алгоритмов анализа графов Флойда и Уоршалла [62—64], или с использованием умножения матриц [65; 66]. Временная сложность алгоритмов, использующих умножение матриц, составляет $\Theta(|V|^3 log(|V|)(T_\otimes + T_\oplus))$, а временная сложность всех остальных упомянутых алгоритмов — $O(|V|^3(T_\otimes + T_\oplus) + |V|^2 T_*)$. Здесь $T_\otimes$, $T_\oplus$ и T_* — временная сложность вычисления операций $s_1 \otimes s_2$, $s_1 \oplus s_2$ и s^* для всех $s, s_1, s_2 \in \mathcal{S}$. Кроме того, существуют некоторые алгоритмы, специализированные для решения частных случаев этой задачи.

Например, рассмотренный алгоритм на листинге 2 является результатом применения подхода *Algebraic Path Problem*, в котором вычисление значений $d_{i,j}$ производятся только для фиксированной начальной вершины i.

Лесли Вэлиант показал, как синтаксический анализ КС-языков может быть проведён с использованием операций над матрицами, модифицированных алгебраическими структурами, схожими с полукольцами, однако без требования ассоциативности операции умножения [22]. Поэтому, возможно удастся использовать идеи, схожие с идеями подхода *Algebraic Path Problem*, для решения задач поиска путей в графе с заданными КС-ограничениями.

1.7 Основные библиотеки линейной алгебры

Одной из причин использования методов линейной алгебры в задачах анализа графов является наличие большого количества высокопроизводительных библиотек линейной алгебры, позволяющих получать хорошие практические результаты для этих задач. Стоит отметить такие библиотеки, как cuBLAS, cuSPARSE, cuBool, m4ri, Scipy и CUSP[1].

При построении алгоритма анализа графов с использованием методов линейной алгебры требуется, во-первых, создать необходимые объекты линейной алгебры (матрицы смежности, вектора) и, во-вторых, использовать библиотечные функции для эффективного вычисления необходимых операций над созданными объектами. Тип данных, используемых в матрицах и векторах зависит от поставленной задачи анализа графов. Часто, для решения задачи достаточно использовать булевы матрицы и вектора (например, для решения задачи достижимости) или такие объекты с более сложным типом данных могут быть выражены с помощью набора булевых матриц и векторов. Например, матрица с конечными множествами в качестве элементов может быть выражена с помощью её булевой декомпозиции. Поэтому наличие в выбранной библиотеке линейной алгебры эффективной реализации операций над булевыми матрицами и векторами крайне важно для многих задач анализа графов. Однако,

[1]Библиотека линейной алгебры CUSP для вычисления операций над разреженными матрицами и решения разреженных систем линейных уравнений на GPU: http://cusplibrary.github.io/ (дата обращения: 14.01.2022).

если поставленную задачу не удаётся свести к вычислению операций над набором булевых матриц и векторов, то необходимо использовать эти объекты линейной алгебры с пользовательским типом данных, а в используемых библиотеках должна быть возможность вычислять операции над ними. Такая ситуация возникает при решении более сложных задач анализа графов, например, при поиске всех путей в графе с некоторыми ограничениями.

Для получения высокопроизводительных реализаций при выборе библиотеки линейной алгебры необходимо учитывать особенности используемых данных и выбранных операций. Например, основные операции над матрицами и векторами могут быть эффективно вычислены с использованием параллельных вычислительных систем (на CPU, GPU, с использованием распределённых вычислений и т.д.). Кроме того, данные на практике разрежены, поэтому важно иметь возможность вычислять операции над разреженными матрицами и векторами в выбранной библиотеке.

Таким образом, можно выделить следующие критерии, которые могут быть использованы при выборе библиотеки линейной алгебры для получения высокопроизводительной реализации алгоритма анализа графов:

- операции над булевыми матрицами/векторами,
- возможность использования пользовательского типа данных и операций над ними,
- использование параллельных вычислений,
- работа с разреженными матрицами/векторами.

Все перечисленные критерии были учтены при создании Айдыном Булуком, Бенджамином Броком (Benjamin Brock), Тимоти Мэттсоном (Timothy Mattson) и другими стандарта GraphBLAS [20], определяющего базовые «строительные блоки» для алгоритмов анализа графов в терминах линейной алгебры. Этот стандарт разрабатывался с 2013 года и был впервые опубликован в мае 2017 года. Для реализации выбранного алгоритма анализа графов можно воспользоваться любой из существующих реализаций стандарта GraphBLAS (SuiteSparse:GraphBLAS [67—69], IBM GraphBLAS[2], GBTL[3] (GraphBLAS

[2]IBM GraphBLAS — реализация стандарта GraphBLAS на языке C++: https://github.com/IBM/ibmgraphblas (дата обращения: 14.01.2022).

[3]GBTL — ещё одна реализация стандарта GraphBLAS на языке C++: https://github.com/cmu-sei/gbtl (дата обращения: 14.01.2022).

Template Library), GraphBLAST[4] [70]). Однако наиболее проработанными и оптимизированными являются реализации SuiteSparse:GraphBLAS на CPU и GraphBLAST — на GPU.

В стандарте GraphBLAS используются разреженные форматы для хранения матриц (CSR, CSC, COO) [20], а также алгебраические структуры, с помощью которых можно модифицировать операции над матрицами и векторами. При применении подхода *Algebraic Path Problem* необходимо использовать полукольца и стандарт GraphBLAS предоставляет такую возможность. Однако стоит отметить, что в стандарте GraphBLAS на используемые алгебраические структуры накладываются более слабые требования, что позволяет использовать структуры более общего вида, чем полукольца. Например, в рамках стандарта возможно создать алгебраическую структуру и с её помощью определить операцию умножения матриц с разными типами данных, что невозможно сделать используя полукольца. Также в стандарте GraphBLAS имеется множество встроенных бинарных операций, из которых могут быть составлены различные алгебраические структуры. Например, может быть составлено полукольцо $\langle \mathbb{R} \cup \{\infty\}, min, +, \infty, 0 \rangle$, которое было использовано для алгоритма Беллмана-Форда в листинге 2. Однако в том случае, если для выбранного алгоритма анализа графов не подходит ни одна из встроенных в стандарт алгебраических структур, то также существует возможность создавать собственные (пользовательские) структуры, описывая тип данных, бинарные операции, а также соответствующие нейтральные элементы.

Характеристики всех перечисленных библиотек линейной алгебры в соответствии с выбранными критериями представлены в табл. 1.

Таким образом, для получения высокопроизводительных реализаций алгоритмов, предлагаемых в данной диссертации, на основе параллельных вычислений на CPU и с возможностью использовать пользовательские типы данных целесообразно использовать реализацию SuiteSparse:GraphBLAS или реализацию IBM GraphBLAS стандарта GraphBLAS. Однако в данной работе сделан выбор в пользу SuiteSparse:GraphBLAS, так как эта реализация является наиболее стабильной и оптимизированной. Для вычисления операций над разреженными булевыми матрицами и векторами на GPU рекомендуется использовать библиотеку cuBool, GraphBLAST или CUSP.

[4]GraphBLAST — реализация стандарта GraphBLAS на GPU с использованием платформы CUDA: https://github.com/gunrock/graphblast (дата обращения: 14.01.2022).

Таблица 1 — Характеристики существующих библиотек линейной алгебры

Библиотека	Булевый тип данных	Пользовательский тип данных	Параллельные вычисления	Разреженные матрицы/ вектора
cuBLAS	-	-	GPU	-
cuSPARSE	-	-	GPU	+
cuBool	+	-	GPU	+
CUSP	+	+	GPU	+
m4ri	+	-	CPU	-
Scipy	+	-	-	+
SuiteSparse:GraphBLAS	+	+	CPU	+
IBM GraphBLAS	+	+	CPU	+
GBTL	+	-	CPU	+
GraphBLAST	+	-	GPU	+

1.8 Выводы

На основе проведённого обзора можно сделать следующие выводы.

- Проблема поиска путей в графе с заданными КС-ограничениями является актуальной в различных областях — в графовых базах данных, биоинформатике, при статическом анализе программ и пр.
- Использование методов линейной алгебры для решения задач анализа графов является перспективным и позволяет существенно улучшить их производительность.
- На сегодняшний день не проводилось исследований о применении методов линейной алгебры для задач поиска путей в графе с заданными КС-ограничениями.

Выполненный обзор также позволяет выявить следующие подходы, технологии и средства, которые целесообразно использовать для решения задач поиска путей в графе с заданными КС-ограничениями.

- Для использования методов линейной алгебры можно использовать идеи подхода *Algebraic Path Problem*, однако необходимо перейти от полуколец к более общим алгебраическим структурам.

- Для задач поиска путей в графах с заданными КС-ограничениями существует возможность использовать представления в виде объектов линейной алгебры не только для графов, но и для КС-ограничений.
- При реализации алгоритмов поиска путей в графах необходимо использовать параллельные вычисления и разреженные форматы для хранения матриц.
- Целесообразно использовать стандарт GraphBLAS, позволяющий определять пользовательские типы данных для матриц и векторов, а также получать высокопроизводительные реализации алгоритмов анализа графов, с использованием операций над матрицами и векторами.
- Для использования параллельных вычислений на CPU можно использовать реализацию SuiteSparse:GraphBLAS стандарта GraphBLAS, а на GPU — библиотеки линейной алгебры cuBool, GraphBLAST и CUSP.

Глава 2. Подход к поиску путей в графе с КС-ограничениями на основе методов линейной алгебры

В этой главе представлен подход к решению класса задач по поиску путей в графе с ограничениями, заданными КС-языком, с помощью методов линейной алгебры. То есть на вход предлагаемый подход получает описание задачи указанного класса, а на выходе выдает алгоритм и некоторые другие артефакты. Несмотря на то, что полностью детерминированную процедуру для создания таких алгоритмов создать не удаётся, видится целесообразным аккумулировать успешный опыт создания ряда алгоритмов этого класса в виде предлагаемого ниже подхода. Этот подход может быть использован как руководство, он также содержит успешные примеры и отсылки к необходимым теоретическим результатам и программным библиотекам. Но он не заменяет акт творческого включения и удачи — это создателям алгоритмов для решения задач из указанного класса придётся обеспечить самостоятельно.

В начале главы предлагается детальное описание предлагаемого подхода. После этого демонстрируется его использование для частного случая задачи поиска путей в графе с заданными КС-ограничениями, в котором ограничения задаются регулярными языками. В конце проведён анализ применимости и ограничений предложенного подхода.

2.1 Описание подхода

Обобщая опыт Лэсли Вэлианта создания матричного алгоритма синтаксического анализа строк, а также подхода *Algebraic Path Problem*, в данном диссертационном исследовании предлагается подход для решения задач поиска путей в графе с КС-ограничениями с помощью методов линейной алгебры.

Схема предлагаемого подхода изображена на рис. 2.1. В качестве фундамента выступают знания из следующих областей: линейная алгебра, теория формальных языков и теория графов. Аппарат линейной алгебры предоставляет: алгоритмы вычисления операций над матрицами и векторами, с помощью которых будет производиться анализ графов; методы решения систем линей-

ных уравнений (такие системы могут возникнуть в матричной форме в процессе построения алгоритма анализа графа); теоретические свойства таких алгоритмов и методов, которые будут во многом определять теоретические свойства результирующего алгоритма; практические результаты в виде готовых высокопроизводительных библиотек линейной алгебры, которые будут использованы в реализации построенного алгоритма. Из теории графов используются: способы представления информации о графах и их связь с объектами линейной алгебры; существующие алгоритмы обхода и поиска путей в графе, которые послужат основой для алгоритмов поиска путей в графе с заданными КС-ограничениями; успешный опыт решения задач анализа графов с помощью методов линейной алгебры, который хорошо описан в книге [13], а для задач поиска путей в графе собран в виде подхода *Algebraic Path Problem* [58; 71—73]; практический опыт применения методов линейной алгебры к задачам анализа графов, который представлен в виде стандарта GraphBLAS [20] и его реализаций. В свою очередь, теория формальных языков предлагает: способы описания КС-языков (формальные грамматики, рекурсивные автоматы), которые будут применены в построенном алгоритме для описания КС-ограничений на пути в графе; нормальные формы КС-грамматик и алгоритмы преобразования грамматик в эти нормальные формы, которые позволят использовать более простые правила вывода для описания ограничений на пути в графе.

На вход предлагаемый подход получает некоторую задачу поиска путей в графе с заданными КС-ограничениями. Такие задачи имеют множество формулировок и могут различаться по следующим измерениям:

- по виду искомой информации о путях в графе (задача достижимости, поиска одного пути, поиска всех путей);
- по множеству пар вершин графа, являющихся концами искомых путей (поиск путей между всеми парами вершин, фиксированное множество начальных вершин, одна начальная и одна конечная вершины и т.д.);
- по наличию дополнительных ограничений на искомые пути в графе в зависимости от решаемой задачи — поиск кратчайших путей, простых путей и т.д.

Стоит отметить, что решаемая задача может иметь практическую или теоретическую направленность. Например, при практической направленности целью применения предлагаемого подхода может быть получение высокопроизводительной реализации для решения поставленной задачи анализа графов.

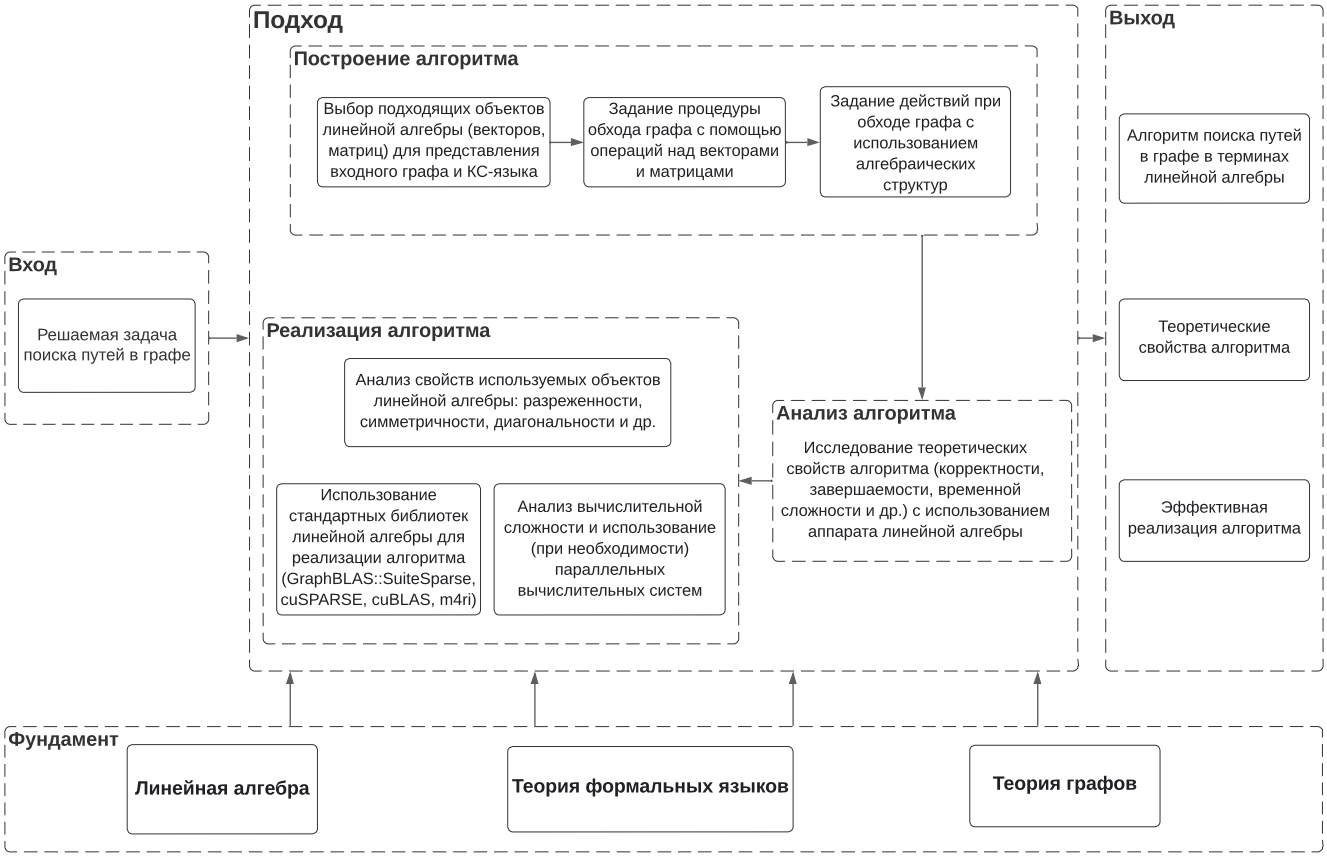


Рисунок 2.1 — Схема подхода к поиску путей в графе с заданными КС-ограничениями с использованием методов линейной алгебры

А при теоретической направленности основным результатом может являться получение алгоритма, который обладает некоторыми уникальными теоретическими свойствами.

Также при постановке решаемой задачи может указываться некоторая дополнительная информация про анализируемые графы и используемые КС-ограничения. Например, может быть известна степень разреженности графов или их особый вид (например, что на вход алгоритму будут подаваться только ацикличные графы). Это, в свою очередь, позволит использовать те или иные специфические структуры для хранения информации о графах, а также операции над этими структурами. Для хранения матриц смежности разреженных графов следует выбирать соответствующие разреженные форматы CSR, CSC, COO. А в случае ацикличности графов — их матрицы смежности будут иметь треугольный вид, что уменьшит сложность вычисления некоторых операций над ними. Например, первый этап метода Гаусса для решения системы линейных уравнений может быть пропущен, так как он приводит матрицу к треугольной форме. А такая операция, как вычисление транзитивно-

го замыкания для треугольной матрицы имеет сложность, равную сложности вычисления одного умножения матриц. Аналогично, могут быть учтены особенности КС-языков, используемых в качестве КС-ограничений, для поиска наиболее подходящей структуры хранения этих ограничений и операций над этой структурой. Так, если перевод формальной грамматики для заданного КС-языка приводит к слишком значительному увеличению размеров этой грамматики, то целесообразно вместо такой трансформации использовать представление КС-языка в виде рекурсивного автомата. Кроме того, если заданные КС-ограничения соответствуют некоторому подклассу класса КС-языков (например, регулярным языкам), то целесообразно использовать соответствующие более простые структуры (например, конечные автоматы) для описания таких ограничений.

Предлагаемый подход делится на следующие части:

- построение алгоритма поиска путей в графе с заданными КС-ограничениями в терминах линейной алгебры;
- анализ теоретических свойств построенного алгоритма и поставленной задачи;
- реализация построенного алгоритма.

Важной частью предлагаемого подхода является само построение алгоритма. Как и при решении других задач анализа графов, ключевым здесь является выбор подходящих объектов линейной алгебры (векторов, матриц) для представления информации о графах. Также подобные объекты можно использовать для представления КС-ограничений путём сведения КС-языков к рекурсивным автоматам [39] с последующим представлением этих автоматов в виде графов. Для представления информации о графах в подавляющем большинстве алгебраических алгоритмов анализа графов используются матрицы смежности (алгоритм обхода в ширину, алгоритм Беллмана-Форда, алгоритм Флойда-Уоршалла) [13]. Однако в некоторых из них также используются вектора (в алгоритме Беллмана-Форда для хранения расстояний от стартовой вершины до всех остальных; в алгоритме обхода в ширину один вектор — для хранения набора вершин, обрабатываемых на текущей итерации, другой — для хранения информации о всех вершинах графа, достижимых из стартовой). Таким образом, для представления графов следует использовать матрицы смежности, а для хранения информации о вершинах графа (достижимость из

стартовой вершины; расстояние от стартовой вершины; набор вершин, инцидентных заданной вершине) следует использовать вектора.

Далее необходимо задать процедуру обхода графа. Как уже указывалось выше, при удачном выборе объектов линейной алгебры на предыдущем шаге эта процедура может быть осуществлёна с помощью операций над выбранными объектами — операциями над матрицами и векторами. Так, например, в алгоритме поиска транзитивного замыкания [71] для обхода графа используется ряд умножений матриц, а именно, возведений матриц во вторую степень. А в алгоритме Беллмана-Форда [13] на каждой итерации рассматриваются пути определённой длины с помощью умножения вектора на матрицу смежности. Следует отметить, что в алгоритмах поиска путей между всеми парами вершин используется умножение матрицы на матрицу, а в случае фиксированной стартовой вершины обход графа осуществляется с использованием умножения вектора на матрицу.

Далее, при успешном выполнении предыдущего шага, необходимо задать дополнительные действия в процессе обхода графа, позволяющие анализировать рассматриваемые пути и проверять их на соответствие заданным КС-ограничениям. Здесь предлагается рассмотреть алгебраические структуры, над которыми будут существовать введенные выше объекты линейной алгебры.

Эти действия могут быть выполнены путём модификации выбранных операций над матрицами и векторами с использованием различных алгебраических структур (полуколец, моноидов и т.д.). Такие модификации с помощью полуколец широко используются для решения задач поиска путей и являются основной идеей подхода *Algebraic Path Problem* [58]. Например, в алгоритме Беллмана-Форда для модификации операции умножения вектора на матрицу используется полукольцо $\langle \mathbb{R} \cup \{\infty\}, min, +, \infty, 0 \rangle$. Также для этого подхода известно много полуколец, которые используются для решения различных задач поиска путей в графе, возникающих в области сетевого анализа [71]. Стоит отметить, что в отличии от подхода *Algebraic Path Problem* предлагаемый подход не ограничивается использованием полуколец для переопределения операций линейной алгебры. Например, Лэсли Вэлиант в своем исследовании о синтаксическом анализе строк для КС-грамматик [22] показал как может быть использована алгебраическая структура, схожая с полукольцом, но без требования ассоциативности операции умножения $\otimes$.

Построенные алгоритмы могут иметь уникальные теоретические свойства, что может являться основной причиной применения предлагаемого подхода для некоторой поставленной задачи. Поэтому ещё одной важной частью подхода является анализ этих свойств. Например, Лэсли Вэлиант в работе [22] создал первый субкубический алгоритм синтаксического анализа строк для КС-грамматик. Это стало возможным благодаря формулированию алгоритма синтаксического анализа с помощью методов линейной алгебры и эффективному вычислению использованных алгебраических операций. Таким образом, формулирование алгоритма с использованием методов линейной алгебры даёт возможность также получить теоретический результат и для самой задачи, ответив на некоторый открытый теоретический вопрос в этой области. Одним из таких вопросов является получение алгоритма для задачи поиска путей в графе с заданными КС-ограничениями, имеющего субкубическую сложность ($(n^{3-\varepsilon}$), где n — количество вершин анализируемого графа, $\varepsilon > 0$).

С практической точки зрения, важной частью предлагаемого подхода является реализация построенного алгоритма. Такие алгоритмы просты в реализации, так как самым трудоёмким является эффективная реализация необходимых операций линейной алгебры, которые уже реализованы в готовых библиотеках. Основные библиотеки линейной алгебры обсуждались в разд. 1.7, а их характеристики представлены в табл. 1.

Так как данные на практике разрежены, то важно хранить матрицы и вектора с использованием разреженных форматов. Например, при выборе матриц смежности входных графов в качестве объектов линейной алгебры используются форматы CSR и CSC для разреженных графов, и формат COO для сильно разреженных графов, когда количество дуг $e \ll n$, где n — количество вершин графа. Кроме того, некоторые операции линейной алгебры над такими объектами могут быть эффективно вычислены параллельно. Поэтому построенные алгоритмы могут быть эффективно реализованы параллельно на CPU, GPU, с использованием распределённых вычислений и т.д. В зависимости от выбранных операций линейной алгебры, формате хранения объектов и выбранной вычислительной системе делается выбор соответствующей библиотеки линейной алгебры или выбранные операции реализуются самостоятельно.

Некоторые задачи анализа графов можно решить с использованием операций над набором булевых матриц и векторов. В таких случаях для эффективного параллельного вычисления операций над разреженны-

ми булевыми матрицами на CPU рекомендуется использовать библиотеку SuiteSparse:GraphBLAS (реализацию стандарта GraphBLAS). Кроме того, в этой библиотеке для матриц и векторов можно определить пользовательский тип данных, с помощью которого могут быть решены более сложные задачи, например, задача поиска всех путей в графе с заданными КС-ограничениями. Кроме того, для получения реализаций на GPU могут быть использованы, например, библиотеки cuSPARSE, cuBool, GraphBLAST и CUSP с реализованными операциями над разреженными матрицами и векторами. Стоит отметить, что в библиотеке cuSPARSE не реализованы специальные функции для вычисления операций над булевыми матрицами и векторами. Поэтому для вычисления операций над булевыми матрицами и векторами с использованием этой библиотеки придётся использовать в качестве типа данных числа с плавающей точкой, что существенно скажется на затратах по памяти и времени работы реализации.

На выход предлагаемый подход даёт возможность получить один или несколько из следующих результатов:

- алгоритм поиска путей в графе с заданными КС-ограничениями, использующий методы линейной алгебры;
- теоретические свойства построенного алгоритма и решаемой задачи поиска путей в графе;
- эффективная реализация построенного алгоритма.

2.2 Пример построения алгоритма

Для демонстрации применения предложенного подхода рассмотрим построение алгоритма для частного случая задачи поиска путей в графе с заданными КС-ограничениями, в котором ограничения задаются строгим подклассом КС-языков — регулярными языками, а сама задача является задачей достижимости. В данной задаче на вход подаётся помеченный граф и регулярный язык, описывающий ограничения на пути в нём.

Сначала необходимо определиться с использованием объектов линейной алгебры для представления входного графа и регулярного языка. Одним из способов задания регулярного языка является построение соответствующего ко-

нечного автомата [36]. В свою очередь входной помеченный граф также может быть представлен в виде конечного автомата, у которого все состояния являются и стартовыми и финальными. Тогда поставленную задачу достижимости с заданными ограничениями можно решить путём нахождения пересечения этих двух конечных автоматов, а также решением задачи достижимости для графового представления этого пересечения. По теореме 1.3.1, вычислять пересечение конечных автоматов можно с использованием произведения Кронекера, применённого к матрицам из булевых декомпозиций матриц смежности для графовых представлений этих автоматов. Таким образом, входные граф и регулряный язык будут представлены в виде набора булевых матриц, а процедурой обхода графа будет являться ряд произведений Кронекера булевых матриц с последующим вычислением транзитивного замыкания.

Пусть конечный автомат F_1 описывает входной граф, а F_2 — входной регулярный язык. На листинге 3 приведён алгоритм, использующий методы линейной алгебры для решения задачи достижимости с заданными ограничениями в виде регулярного языка. В строке 7 алгоритм производит обход графа с использованием произведения Кронекера, в результате которого будет построена матрица переходов конечного автомата F, являющегося пересечением конечных автоматов F_1 и F_2. Здесь булевы матрицы M_i^a хранят в себе информацию о переходах с символом a в автомате F_i. Затем в строке 8 вычисляется матрица M^*, которая содержит информацию о путях во входном графе, которые соответствуют заданным ограничениям. Для этого используется функция *transitiveClosure* и операция $\bigvee$ поэлементного сложения матриц, определённая над булевым полукольцом $\langle\{0,1\},\vee,\wedge,0,1\rangle$. Стоит отметить, что функция *transitiveClosure* также может быть реализована с использованием методов линейной алгебры, а именно через серию возведений передаваемой булевой матрицы во вторую степень. И, наконец, в строках 9–12 вычисляется множество *result*, являющееся ответом на задачу достижимости в графе с ограничениями в виде регулярного языка. При этом используется тот факт, что наличие в ячейке $M^*[(i,s_1),(j,s_2)]$ значения 1 эквивалентно существованию пути π_1 во входном графе из вершины i в вершину j, и пути π_2 в графовом представлении конечного автомата F_2 из вершины s_1 в вершину s_2, где $\lambda(\pi_1)=\lambda(\pi_2)$. Это означает, что для любой пары вершин (i,j) входного графа, вершина j достижима из вершины i хотя бы одним путём, удовлетворяющим входным ограничениям тогда и только тогда, когда $M^*[(i,s_1),(j,s_2)]=1$ для

вершины s_1, соответствующей начальному состоянию конечного автомата F_2, и вершины s_2, соответствующей одному из его конечных состояний. Таким образом, построенный алгоритм решает поставленную задачу анализа графов.

Листинг 3 Алгоритм достижимости в графе с заданными ограничениями в виде регулярного языка

```
1:  function AlgebraicRegularPathQuerying(F_1, F_2)
2:      M_1 ← булева декомпозиция матрицы смежности для F_1
3:      M_2 ← булева декомпозиция матрицы смежности для F_2
4:      q_s ← начальное состояние конечного автомата F_2, описывающего регулярный язык
5:      Q_f ← множество конечных состояний автомата F_2
6:      Σ ← множество общих символов на переходах в автоматах F_1 и F_2
7:      M ← {M_1^a × M_2^a | a ∈ Σ}                    ▷ Произведение Кронекера
8:      M* ← transitiveClosure(⋁_{M∈M} M)
9:      result ← ∅
10:     for i, j, s | M*[(i, q_s), (j, s)] = 1 do
11:         if s ∈ Q_f then
12:             result ← result ∪ {(i, j)}
13:     return result
```

Для решения задачи достижимости была использована операция произведения Кронекера булевых матриц, основанная на стандартных логических операций из булевого полукольца $\langle\{0,1\},\vee,\wedge,0,1\rangle$. В данном случае не требуется задавать дополнительные действия при обходе графа с помощью изменения этой алгебраической структуры. Однако такие действия могут понадобиться при решении других задач поиска путей в графе, например, при поиске всех путей в графе с заданными КС-ограничениями.

Таким образом, было показано, как может быть построен алгоритм с использованием методов линейной алгебры для решения задачи достижимости с заданными ограничениями в виде регулярного языка.

2.3 О применимости и ограничениях подхода

Далеко не каждый алгоритм анализа графов и, в частности, алгоритм поиска путей в графе, удаётся сформулировать в терминах линейной алгебры. Например, было сделано много попыток найти такую формулировку для алгоритма поиска в глубину — этот алгоритм является одним из фундаментальных алгоритмов анализа графов. Однако недавно это удалось сделать лишь для частных случаев графов (двоичного дерева, ациклического графа) [21]. В свою очередь, алгоритм поиска в ширину легко может быть сформулирован в терминах линейной алгебры и такая формулировка была давно найдена [13].

Таким образом, при применении предложенного подхода на этапе построения алгоритма могут возникнуть следующие проблемы:

- сложность задания процедуры обхода графа с использованием операций над выбранными объектами линейной алгебры для представления входного графа и КС-языка;
- сложность поиска алгебраических структур для задания действий при обходе графа, необходимых для анализа рассматриваемых путей в рамках поставленной задачи.

При неудачном решении этих проблем необходимо вернуться к выбору объектов линейной алгебры. Такой возврат также необходим при выявлении проблем на более поздних этапах подхода. Например, на этапе анализа теоретических свойств построенного алгоритма может быть выявлена неудовлетворительная временная сложность алгоритма. В то же время на этапе реализации может возникнуть сложность написания собственной реализации построенного алгоритма или следующие проблемы, связанные с использованием стандартных библиотек линейной алгебры:

- свойства входных данных и использованных объектов линейной алгебры для их представления не позволяют выбрать подходящую стандартную библиотеку;
- при использовании подходящей стандартной библиотеки выявлены неудовлетворительные время работы реализации алгоритма или объём потребляемой памяти.

Стоит также отметить, что не все этапы подхода являются обязательными. Например, при практической направленности решаемой задачи может

быть пропущен этап анализа теоретических свойств алгоритма, а при выборе используемых объектов линейной алгебры и операций над ними необходимо руководствоваться возможностями существующих библиотек линейной алгебры. С другой стороны, при теоретической направленности решаемой задачи может отсутствовать этап реализации построенного алгоритма, а на этапе построения алгоритма необходимо выбрать подходящие объекты линейной алгебры и операции над ними для получения наилучших теоретических свойств алгоритма.

2.4 Выводы

Таким образом, в данной главе представлен подход к поиску путей в произвольном графе с заданными произвольными КС-ограничениями с использованием методов линейной алгебры. Новизна подхода заключается в следующем.

1. Имеющиеся на сегодняшний день подходы к поиску путей в графах с заданными КС-ограничениями либо не используют методы линейной алгебры, либо предназначены только для частного случая КС-ограничений и/или специализированных графов.
2. Предложенный подход позволяет использовать широкий класс оптимизаций операций линейной алгебры для эффективного анализа больших графов.
3. Предложенный подход позволяет строить алгоритмы, которые просты в реализации, переносимы, позволяют легко и эффективно использовать параллельные вычисления за счёт существующих библиотек линейной алгебры.

Глава 3. Алгоритм поиска путей в графе с заданными КС-ограничениями с использованием операций умножения матриц

В данной главе изложен алгоритм, полученный в результате применения подхода из предыдущей главы для решения задачи достижимости, поиска одного и поиска всех путей в графе с заданными КС-ограничениями с использованием операций умножения матриц. Также сформулированы и доказаны утверждения о корректности и временной сложности полученного алгоритма. Кроме того, приведены детали реализаций алгоритма, а также его работа продемонстрирована на примере.

3.1 Построение алгоритма

В данном разделе изложен процесс построения алгоритма поиска путей в графе с заданными КС-ограничениями с использованием операций умножения матриц.

Пусть дан входной помеченный граф $\mathcal{G} = \langle V, E, L \rangle$ и входной КС-язык в качестве ограничений на пути в нём. Сперва необходимо выбрать объекты линейной алгебры для представления информации о графе. Применимость таких объектов линейной алгебры, как матрицы в задачах анализа графов давно известна. Поэтому для представления входного графа будем использовать его матрицу смежности, в ячейках (i, j) которой будет содержаться информация о дугах между вершиной i и вершиной j. Входной КС-язык будем описывать КС-грамматикой $G = \langle \Sigma, N, P, S \rangle$ в ослабленной нормальной форме Хомского, которая, как мы увидим, будет удобна для дальнейшего построения алгебраических структур с операциями, учитывающими заданные КС-ограничения в процессе поиска путей в графе. Затем матрицу смежности необходимо инициализировать, включив в неё информацию о заданных ограничениях. Для описания связи найденных путей в графе с заданными КС-ограничениями будут использоваться нетерминальные символы грамматики G. Таким образом, в процессе работы предлагаемый алгоритм для всех рассматриваемых путей π будет выяснять из каких нетерминалов A выводится строка $\lambda(\pi)$. Поэтому для

инициализации матрицы смежности входного графа будут рассмотрены все дуги с метками $a \in \Sigma \cap L$ и соответствующие им правила вида $A \to a$. Таким образом, будут рассмотрены все пути графа длины 1, но также необходимо рассмотреть пустые пути, при условии наличия в грамматике G правил вида $A \to \varepsilon$. Поэтому для каждого такого правила и для каждой вершины i графа $\mathcal{G}$ в матрицу будет записана информация о наличии пустого пути π из вершины i в вершину i такого, что строка $\lambda(\pi)$ выводится из нетерминала A.

Далее опишем процедуру обхода графа, основанную на вычислении транзитивного замыкания инициализированной матрицы. Такое транзитивное замыкание может быть вычислено с использованием известной техники [71], в которой производится серия умножений матрицы смежности на себя. Это позволит обойти граф и рассмотреть все необходимые для анализа графа пути. Остаётся лишь переопределить операцию умножения таких матриц, чтобы, во-первых, в процессе обхода графа рассматриваемые пути проверялись на соответствие входным КС-ограничениям, а, во-вторых, вычислялась вся необходимая информация для решения поставленной задачи достижимости, поиска одного или поиска всех путей в графе. Для проверки путей на соответствие входным КС-ограничениям будут использованы правила вида $A \to BC$, где $A, B, C \in N$. В правой части таких правил имеется лишь конкатенация двух нетерминалов, что в процессе вывода строки в грамматике соответствует конкатенации двух подстрок. Аналогично, в процессе анализа графа правила такого вида соответствуют конкатенации двух коротких путей $i\pi_1 k$ и $k\pi_2 j$, где $B \Rightarrow_G \lambda(\pi_1)$ и $C \Rightarrow_G \lambda(\pi_2)$. В таком случае, для пути $i\pi j$, полученного в результате такой конкатенации мы можем утверждать, что $A \Rightarrow_G \lambda(\pi)$. Таким образом, процесс обхода графа $\mathcal{G}$ будет неразрывно связан с процессом вывода строк в грамматике G, образованных рассмотренными путями графа. И, наконец, чтобы вычислялась вся необходимая информация для решения поставленной задачи поиска путей в графе, построим алгебраическую структуру $\langle \mathit{MatrixElements}, \oplus, \otimes, \perp \rangle$, где:

- *MatrixElements* — множество, содержащее в себе все возможные значения элементов рассматриваемых матриц;
- $\oplus$ — операция сложения элементов матриц, которая будет использоваться при агрегации информации о нескольких путях в графе между одними и теми же вершинами;

- $\otimes$ — операция умножения элементов матриц, которая будет использоваться при агрегации информации о двух путях, которые могут быть сконкатенированы в один более длинный путь;
- $\perp$ — нейтральный по сложению элемент, который будет обозначать отсутствие искомых путей для конкретной пары вершин.

Тогда используя построенную алгебраическую структуру определим операцию умножения матриц $a \cdot b = c$, где a и b — матрицы подходящих размеров с элементами из множества *MatrixElements*, как

$$c_{i,j} = \bigoplus_{k=1}^{n} a[i,k] \otimes b[k,j].$$

Кроме того, для агрегации информации о путях из двух матриц одинакового размера будем использовать операцию $\bigoplus$ поэлементного сложения элементов этих двух матриц, определённую над той же алгебраической структурой. Также на этапе инициализации необходимо определить элементы, используемые для описания информации о путях длины 0 и 1. Так как значения этих элементов зависят от поставленной задачи поиска путей в графе, то обозначим их, как $\alpha^0_{i,j}, \alpha^1_{i,j} \in$ *MatrixElements*, для каждой пары вершин (i,j).

Таким образом, на листинге 4 представлен алгоритм поиска путей в графе с заданными КС-ограничениями, использующий операции умножения матриц. Представленный алгоритм принимает на вход граф и КС-ограничения уже выраженные в виде КС-грамматики в ослабленной нормальной форме Хомского. Стоит отметить, что вместо одной матрицы информация о путях в графе хранится в множестве матриц T, состоящем из $|N|$ матриц T^A по одной на каждый нетерминальный символ $A \in N$. В таком случае в ячейку $T^A[i,j]$ записывается информация о найденных путях в графе из вершины i в вершину j, образующих строки, выводимые из нетерминала A в грамматике G. Данная идея схожа с идеей использования булевой декомпозиции матрицы смежности, однако получаемые матрицы T^A будут булевыми только для задачи достижимости. В итоге, предложенный алгоритм решает поставленную задачу поиска путей в графе $\mathcal{G}$ с заданными КС-ограничениями, так как вся необходимая информация о путях из вершины i в вершину j, удовлетворяющих заданным КС-ограничениям в виде грамматики $G = \langle \Sigma, N, P, S \rangle$, будет записана в ячейку $T^S[i,j]$.

Далее представим различные алгебраические структуры, позволяющие переопределить операции над матрицами в алгоритме, представленном на ли-

Листинг 4 Алгоритм поиска путей в графе с заданными КС-ограничениями, использующий операции умножения матриц

```
1: function MatrixBasedCFPQ(𝒢 = ⟨V, E, L⟩, G = ⟨N, Σ, P, S⟩)
2:     n ← |V|
3:     T ← {T^A | A ∈ N, где T^A — матрица размера n × n со всеми элементами
       равными ⊥ }
4:     for all (i, x, j) ∈ E, A | A → x ∈ P do
5:         T^A[i, j] ← α^1_{i,j}        ▷ Инициализация матриц для правил вида A → x
6:     for A | A → ε ∈ P do
7:         for all i ∈ {0, ..., n − 1} do
8:             T^A[i, i] ← α^0_{i,i}    ▷ Инициализация матриц для правил вида A → ε
9:     while любая матрица из T меняется do
10:        for all A → BC ∈ P, где T^B или T^C изменились do
11:            T^A ← T^A ⨁ (T^B · T^C)
12:    return T
```

стинге 4, для решения задачи достижимости, поиска одного пути или поиска всех путей в графе с заданными КС-ограничениями.

Задача достижимости. Так как для решения задачи достижимости необходима лишь информация о наличии путей, образующих из меток своих дуг определённые слова, то в ячейках матрицы смежности могут содержаться только булевы значения 0 или 1. В таком случае в ячейку $T^A[i, j]$ записывается значение 1, если существует путь из вершины i в вершину j, образующий строку, выводимую из нетерминала A в грамматике G, и значение 0 в противном случае. В процессе инициализации матриц в ячейки $T^A[i, j]$ будут записываться значения $\alpha^0_{i,j}$ и $\alpha^1_{i,j}$ при обнаружении соответствующего пути длины 0 или 1. А так как для поставленной задачи наличие пути записывается в соответствующую ячейку с помощью значения 1, то $\alpha^0_{i,j} = \alpha^1_{i,j} = 1$. Таким образом, алгоритм, представленный на листинге 4 решает задачу достижимости в графе с заданными КС-ограничениями при использовании алгебраической структуры $\langle \{0, 1\}, \vee, \wedge, 0 \rangle$ с логическими операциями дизъюнкции и конъюнкции. В графе $\mathcal{G}$ существует путь из вершины i в вершину j, удовлетворяющий заданным КС-ограничениям, только если $T^S[i, j] = 1$.

Задача поиска одного пути. Для решения задачи поиска одного пути в графе с заданными КС-ограничениями, необходимо для каждой пары вершин (i, j) иметь возможность построить хотя бы одни путь из вершины i в вершину j, удовлетворяющий заданным ограничениям, если такие пути существуют. Для этого добавим в ячейки матриц дополнительную информацию о найденных путях в графе и построим новую алгебраическую структуру для модификации операций над этими матрицами. Будем использовать матрицы, в ячейках которых записана информация о найденных путях в виде четвёрок (*left*, *right*, *middle*, *height*), где *left* и *right* — конечные вершины найденного пути, *middle* — одна из промежуточных вершин найденного пути π со строкой $\lambda(\pi)$, выводимой из нетерминала A в грамматике G, и *height* — минимальная высота дерева вывода строки $\lambda(\pi)$ из нетерминала A. В случае, когда для определенной пары вершин (i, j) таких путей не обнаружено, то будем использовать четвёрку $\perp = (0, 0, 0, 0)$.

Для конкретной пары вершин (i, j) и нетерминала A в предложенном алгоритме будет рассматриваться путь, образующий строку, выводимую из нетерминала A и имеющую минимальную высоту дерева вывода. Кроме того, будут рассматриваться конкатенации двух таких путей для различных троек (A, i, j). Поэтому в предложенном алгоритме будут использоваться только четвёрки (*left*, *right*, *middle*, *height*) с $height \leqslant h + 1$, где h — максимальная из таких минимальных высот деревьев вывода для различных троек (A, i, j). Обозначим множество всех возможных таких четвёрок, включая $\perp$, как *PathIndex*, и построим алгебраическую структуру, используя это множество как носитель. Нейтральным по сложению элементом в такой структуре будет являться четвёрка $\perp$. В таком случае, изначально все матрицы будут инициализированы этим нейтральным элементом. Далее необходимо определить операции умножения и сложения для этой структуры.

Так как в процессе обхода графа с использованием операции умножения матриц и правил грамматики вида $A \to BC$, рассматриваются более длинные пути $i\pi j$, являющиеся конкатенацией двух коротких путей $i\pi_1 k$ и $k\pi_2 j$, то в качестве промежуточной вершины *middle* длинного пути удобно выбрать вершину k. Тогда операция умножения $\otimes$ для $PI_1, PI_2 \in PathIndex$, может быть определена следующим образом.

$$PI_1 \otimes PI_2 = \begin{cases} (PI_1.left, PI_2.right, PI_1.right, max(PI_1.height, PI_2.height) + 1), \\ \qquad \text{если } PI_1 \neq \bot \neq PI_2 \\ \bot, \qquad \text{иначе} \end{cases}$$

Кроме того, в процессе анализа графа для одной и той же пары вершин могут быть найдены несколько путей, удовлетворяющих заданным КС-ограничениям. Такие пути агрегируются с использованием операции сложения $\oplus$ для $PI_1, PI_2 \in PathIndex$, которая может быть определена следующим образом.

$$PI_1 \oplus PI_2 = \begin{cases} PI_1, \qquad \text{если } PI_1 \neq \bot \neq PI_2 \text{ и} \\ \qquad (PI_1.height, PI_1.middle) \leqslant (PI_2.height, PI_2.middle) \\ PI_1, \qquad \text{если } PI_2 = \bot \\ PI_2, \qquad \text{иначе} \end{cases}$$

В представленном определении использовался лексикографический порядок над парами целых чисел $(PI.height, PI.middle)$ для $PI \in PathIndex$. Таким образом, при использовании операции $\oplus$ в ячейках матрицы T^A будет содержаться информация о найденных путях, образующих строки с минимальными высотами деревьев вывода из нетерминала A в грамматике G. Далее определим значения элементов $\alpha^0_{i,j}, \alpha^1_{i,j} \in PathIndex$. При обнаружении пути длины 1, выводимого из нетермниала A_k в соответствующую ячейку матрицы T^{A_k} будет добавляться информация об этом пути в виде четвёрки $\alpha^1_{i,j} = (i, j, i, 1)$. В свою очередь для путей длины 0 в соответствующие ячейки добавляются четвёрки $\alpha^0_{i,i} = (i, i, i, 1)$. В данных случаях в качестве промежуточных вершины выбраны начальные вершины путей длины 0 и 1. Таким образом, алгоритм, представленный на листинге 4 позволяет решить задачу поиска одного пути в графе с заданными КС-ограничениями при использовании полукольца $\langle PathIndex, \oplus, \otimes, \bot \rangle$ с определенными для данной задачи операциями. Результатом работы алгоритма является множество матриц T^A для всех нетерминальных символов $A \in N$, в ячейках (i, j) которых содержится информация об одном найденном пути из вершины i в вершину j, образующем строку, выводимую из нетерминала A с минимальной высотой дерева вывода. В случае, если для пары вершин (i, j) таких путей не существует, то $T^A[i, j] = \bot$.

Однако кроме построения множества матриц T, содержащих информацию о найденных путях, необходимо восстановить путь $i\pi j$ для любых i, j и нетерминала A таких, что $A \Rightarrow_G \lambda(\pi)$, если такой путь существует. Поэтому на листинге 5 представлен алгоритм восстановления одного пути, соответствующего заданным КС-ограничениям, для любой пары вершин (i, j) и нетерминала A. Представленный алгоритм восстанавливает путь, соответствующий строке с минимальной высотой дерева вывода. Алгоритм возвращает пустой путь π_ε только когда $i = j$ и $A \to \varepsilon \in P$. Необходимо заметить, что если $T^A[i, j] = \bot$, то предложенный алгоритм возвращает специальный путь $\pi_\varnothing$, обозначающий отсутствие путей соответствующих заданным КС-ограничениям.

Листинг 5 Алгоритм восстановления одного пути в графе с заданными КС-ограничениями

```
1: function ExtractSinglePath(i, j, A, T = {T^{A_i}}, G = ⟨N, Σ, P, S⟩)
2:     index ← T^A_{i,j}
3:     if index = ⊥ then
4:         return π_∅                          ▷ Такого пути не существует
5:     if index.height = 1 then
6:         if (i = j) ∧ (A → ε ∈ P) then
7:             return π_ε                      ▷ Возвращаем пустой путь
8:         for all x | (i,x,j) ∈ E do
9:             if A → x ∈ P then
10:                return [(i,x,j)]            ▷ Возвращаем путь длины 1
11:    for all A → BC ∈ P do
12:        index_B ← T^B[i, index.middle]
13:        index_C ← T^C[index.middle, j]
14:        if (index_B ≠ ⊥) ∧ (index_C ≠ ⊥) then
15:            maxH ← max(index_B.height, index_C.height)
16:            if index.height = maxH + 1 then
17:                π_1 ← ExtractSinglePath(i, index.middle, B, T, G)
18:                π_2 ← ExtractSinglePath(index.middle, j, C, T, G)
19:                return π_1 + π_2            ▷ Возвращаем конкатенацию двух путей
```

Таким образом, алгоритмы представленные на листингах 4 и 5 позволяют решить задачу поиска одного пути в графе с заданными КС-ограничениями с использованием операций умножения матриц.

Задача поиска всех путей. Для решения задачи поиска всех путей в графе с заданными КС-ограничениями, необходимо для каждой пары вершин (i, j) иметь возможность построить все пути из вершины i в вершину j, удовлетворяющие заданным ограничениям. Для этого добавим в ячейки матриц дополнительную информацию о всех найденных путях в графе и построим новую алгебраическую структуру для модификации операций над этими матрицами. В качестве такой информации возьмём множество троек $(left, right, middles)$, где $left$ и $right$ — начальная и конечная вершины найденных путей, а $middles$ — множество некоторых промежуточных вершин этих путей. В процессе обхода графа в $T^A[i, j].middles$ будут записываться все промежуточные вершины k рассматриваемых путей $i\pi j$, образованных конкатенацией двух более коротких путей $i\pi_1 k$ и $k\pi_2 j$. В случае, если путь имеет длину 0 или 1 и не имеет промежуточных вершин, то будем добавлять в соответствующую ячейку (i, j) элементы $\alpha^0_{i,j} = \alpha^1_{i,j} = (i, j, \{n\})$ со специальным значением $n = |V|$. Такое значение отличается от номера любой вершины рассматриваемого графа, что позволит отдельно рассматривать случай, когда найденный путь не имеет промежуточных вершин. А в случае отсутствия путей, соответствующих заданным ограничениям для некоторой пары вершин, будем использовать тройку $\perp = (0, 0, \varnothing)$. Обозначим множество всех возможных значений таких ячеек, включая $\perp$, как $AllPathIndex$. Теперь построим алгебраическую структуру над элементами множества $AllPathIndex$, которое позволит модифицировать операции умножения и сложения матриц для решения задачи поиска всех путей в графе с заданными КС-ограничениями. Нейтральным элементом по сложению в этой структуре будет элемент $\perp$.

Для построения алгебраической структуры сначала определим операцию умножения $\otimes$ для $API_1, API_2 \in AllPathIndex$ следующим образом.

$$API_1 \otimes API_2 = \begin{cases} (API_1.left, API_2.right, \{API_1.right\}), \text{если } API_1 \neq \perp \neq API_2 \\ \perp, \qquad \text{иначе} \end{cases}$$

При обнаружении нескольких путей в графе, удовлетворяющих заданным КС-ограничениям, для одной и той же пары вершин необходимо записать информацию обо всех найденных путях. Поэтому определим операцию сложения $\oplus$ для $API_1, API_2 \in AllPathIndex$ следующим образом.

$$API_1 \oplus API_2 = \begin{cases} (API_1.left, API_1.right, \\ API_1.middles \cup API_2.middles), & \text{если } API_1 \neq \bot \\ API_2, & \text{иначе} \end{cases}$$

Таким образом, алгоритм, представленный на листинге 4 позволяет решить задачу поиска всех путей в графе с заданными КС-ограничениями при использовании алгебраической структуры $\langle AllPathIndex, \oplus, \otimes, \bot \rangle$ с определенными для данной задачи операциями. Результатом работы алгоритма является множество матриц T^A для всех нетерминальных символов $A \in N$, в ячейках (i, j) которых содержится информация обо всех путях из вершины i в вершину j, образующих строки, выводимые из нетерминала A. В случае, если для пары вершин (i, j) таких путей не существует, то $T^A[i, j] = \bot$.

После построения множества матриц T, содержащих информацию обо всех найденных путях, необходимо иметь возможность восстановить все пути $i\pi j$ для любых i, j и нетерминала A таких, что $A \Rightarrow_G \lambda(\pi)$. Поэтому на листинге 6 представлен алгоритм восстановления всех путей, соответствующих заданным КС-ограничениям, для любой пары вершин (i, j) и нетерминала A. Алгоритм возвращает пустой путь π_ε только когда $i = j$ и $A \to \varepsilon \in P$.

Таким образом, алгоритмы представленные на листингах 4 и 6 позволяют решить задачу поиска всех путей в графе с заданными КС-ограничениями с использованием операций умножения матриц.

3.2 Корректность алгоритма

В данном разделе сформулированы и доказаны утверждения о корректности и завершаемости изложенного алгоритма для рассмотренных задач поиска путей в графе с заданными КС-ограничениями.

Листинг 6 Алгоритм восстановления всех путей в графе с заданными КС-ограничениями

```
1: function ExtractAllPaths(i, j, A, T = {T^{A_k} | A_k ∈ N}, G = ⟨N, Σ, P, S⟩)
2:     index ← T^A_{i,j}
3:     if index = ⊥ then
4:         return ∅                                  ▷ Таких путей не существует
5:     n ← количество строк матрицы T^A
6:     resultPaths ← ∅
7:     for all m ∈ index.middles do
8:         if m = n then                             ▷ Добавляем путь длины 0 или 1
9:             for all x | A → x ∈ P do
10:                if (i, x, j) ∈ E then
11:                    resultPaths ← resultPaths ∪ {((i, x, j))}
12:            if (i = j) ∧ (A → ε ∈ P) then
13:                resultPaths ← resultPaths ∪ {π_ε}
14:        else      ▷ Добавляем конкатенацию путей из i в m и путей из m в j
15:            for all A → BC ∈ P do
16:                index_B ← T^B[i, m]
17:                index_C ← T^C[m, j]
18:                if (index_B ≠ ⊥) ∧ (index_C ≠ ⊥) then
19:                    lPaths ← ExtractAllPaths(i, m, B, T, G)
20:                    rPaths ← ExtractAllPaths(m, j, C, T, G)
21:                    resultPaths ← resultPaths ∪ lPaths · rPaths
22:    return resultPaths
```

Пусть дана одна из трёх задач поиска путей в графе с заданными КС-ограничениями. Также пусть для поставленной задачи построена алгебраическая структура $\langle MatrixElements, \oplus, \otimes, \perp \rangle$, переопределяющая операции над матрицами в алгоритме, представленном на листинге 4. Кроме того, пусть для любой пары вершин (i, j) входного графа $\mathcal{G}$ в носителе этой структуры выделены элементы $\alpha^0_{i,j}$ и $\alpha^1_{i,j}$, используемые для инициализации матриц этого алгоритма. Тогда сначала докажем, что если построенная структура обладает некоторыми свойствами, то получившийся алгоритм поиска путей в графе завершается за конечно число шагов и корректно решает поставленную задачу. А затем покажем, что предложенные в предыдущем разделе алгебраические структуры для задач достижимости, поиска одного и поиска всех путей в графе с заданными КС-ограничениями такими свойствами обладают.

Завершаемость. Для доказательства завершаемости алгоритма, представленного на листинге 4, сперва введём обозначения для промежуточных матриц из множества T, получающихся в процессе работы этого алгоритма. Для любой пары вершин (i, j) и для любого нетерминала $A \in N$, будем использовать обозначение $T^{A,0}[i, j]$ для значения в ячейке $T^A[i, j]$ после инициализации матриц в строках 2–8 алгоритма, а $T^{A,k}[i, j]$ — для значения в ячейке $T^A[i, j]$ после k исполнений цикла в строках 9–11, для $k \geqslant 1$.

Тогда докажем, что если на элементах использованной конечной алгебраической структуры $\langle MatrixElements, \oplus, \otimes, \perp \rangle$ может быть задано некоторое отношение частичного порядка $\preceq$, то предложенный алгоритм завершается.

Теорема 3.2.1 (Завершаемость алгоритма)**.** Если на элементах использованной алгебраической структуры $\langle MatrixElements, \oplus, \otimes, \perp \rangle$ может быть задано отношение частичного порядка $\preceq$ такое, что для любой пары вершин (i, j), любого нетерминала $A \in N$ и любого $k \geqslant 1$, $T^{A,k-1}[i, j] \preceq T^{A,k}[i, j]$ и носитель $MatrixElements$ — конечен, то алгоритм, представленный на листинге 4 завершается за конечное число шагов.

Доказательство. На очередной итерации цикла в строках 9–11 алгоритма, значения в ячейках $T^A[i, j]$ либо могут не изменяться, либо изменяться в результате выполнения операций $T^A \leftarrow T^A \bigoplus (T^B \cdot T^C)$ в строке 11. Алгоритм продолжает свою работу пока значение хотя бы в одной ячейке изменяется. По условию теоремы, значения в ячейках матриц от итерации к итерации монотонно возрастают относительно заданного частичного порядка $\preceq$. Таким образом, в силу

конечности носителя *MatrixElements* на некоторой итерации матрицы перестанут изменяться и алгоритм завершит свою работу. □

Корректность. Для доказательства корректности алгоритма, представленного на листинге 4, введём обозначения, описывающие связь рассматриваемых путей в графе и значений в ячейках $T^{A,k}[i,j]$ промежуточных матриц, получающихся в процессе работы этого алгоритма. Для любой пары вершин (i,j) и для любого нетерминала $A \in N$, будем использовать обозначение $\mathcal{P}_{\mathcal{G},k}(i,j,A)$ для множества всех путей $i\pi j$ графа $\mathcal{G}$ таких, что существует дерево вывода минимальной высоты $h \leqslant k$ для строки $\lambda(\pi)$ из нетерминала A грамматики G. Также будем говорить, что некоторая информация позволяет корректно решить задачу поиска путей с заданными КС-ограничениями для множества путей $\mathcal{P}$ из вершины i в вершину j, если эта информация:

- позволяет ответить на вопрос существования хотя бы одного пути $\pi \in \mathcal{P}$, удовлетворяющего заданным ограничениям, для задачи достижимости;
- позволяет построить хотя бы один путь $\pi \in \mathcal{P}$, удовлетворяющий заданным ограничениям, если такой существует, для задачи поиска одного пути;
- позволяет построить любое конечное количество путей $\pi \in \mathcal{P}$, удовлетворяющих заданным ограничениям, для задачи поиска всех путей.

Тогда докажем, что если операции использованной алгебраической структуры $\langle \mathit{MatrixElements}, \oplus, \otimes, \perp \rangle$ и её элементы обладают некоторыми свойствами, то предложенный алгоритм корректно решает поставленную задачу.

Лемма 3.2.2 (Корректность алгоритма)**.** Пусть $\mathcal{G} = \langle V, E, L \rangle$ — входной граф и $G = \langle N, \Sigma, P, S \rangle$ — входная КС-грамматика для алгоритма, представленного на листинге 4. Также пусть для использованной алгебраической структуры $\langle \mathit{MatrixElements}, \oplus, \otimes, \perp \rangle$ и для любых элементов $\alpha_1, \alpha_2 \in \mathit{MatrixElements}$ справедливы следующие свойства:

- алгебраическая структура $\langle \mathit{MatrixElements}, \oplus, \otimes, \perp \rangle$ является полукольцом без требования ассоциативности операции $\otimes$;
- выделенные элементы $\alpha^l_{i,j} \in \mathit{MatrixElements}$, использованные в строках 4–8 алгоритма, позволяют корректно решить поставленную задачу для множеств путей графа $\mathcal{G}$ из вершины i в вершину j длины l, где $l \in \{0, 1\}$;

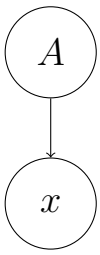


Рисунок 3.1 — Дерево вывода минимальной высоты $h = 1$ для строки $x = \lambda(\pi)$, где $x \in \Sigma \cup \{\varepsilon\}$

- если элементы α_1 и α_2 позволяют корректно решить поставленную задачу для множеств путей $\mathcal{P}_1$ и $\mathcal{P}_2$ из вершины i в вершину j, то элемент $\alpha_1 \oplus \alpha_2$ позволяет корректно решить поставленную задачу для множества путей $\mathcal{P}_1 \cup \mathcal{P}_2$;
- если элементы α_1 и α_2 позволяют корректно решить поставленную задачу для множества путей $\mathcal{P}_1$ из вершины i в вершину k и для множества $\mathcal{P}_2$ путей из вершины k в вершину j, то элемент $\alpha_1 \otimes \alpha_2$ позволяет корректно решить поставленную задачу для множества путей $\mathcal{P}_1 \cdot \mathcal{P}_2$ всех возможных конкатенаций путей из этих множеств.

Тогда для любой пары вершин (i, j) графа $\mathcal{G}$, для любого нетерминала $A \in N$ и для любого $k \geqslant 0$ значение в ячейке $T^{A,k}[i, j]$ позволяет корректно решить поставленную задачу для множества путей $\mathcal{P}_{\mathcal{G},k+1}(i, j, A)$.

Доказательство. (Доказательство методом математической индукции)

База: Покажем, что утверждение леммы справедливо для $k = 0$. Для грамматик в ослабленной нормальной форме Хомского, деревья вывода с высотой 1 имеют лишь строки длины 1 или пустая строка ε. Таким образом, множество $\mathcal{P}_{\mathcal{G},1}(i, j, A)$ содержит только пути длины 0 или 1. Для всех таких путей и только для них существует дерево вывода минимальной высоты $h = k+1 = 1$, показанное на рисунке 3.1. Рассмотрение таких путей происходит в строках 4–8 алгоритма. Поэтому для любой пары вершин (i, j) и любого нетерминала $A \in N$, $T^{A,0}[i, j] \neq \perp$ тогда и только тогда, когда либо существует путь $i\pi j$ длины 1, который содержит единственную дугу $(i, x, j) \in E$ и $(A \to x) \in P$, либо $i = j$ и $(A \to \varepsilon) \in P$. В случае наличия пути $\pi \in \mathcal{P}_{\mathcal{G},1}(i, j, A)$, в ячейку $T^{A,0}[i, j]$ будут записаны соответствующие элементы $\alpha^0_{i,j}$ или $\alpha^1_{i,j}$. А по условию леммы выделенные элементы $\alpha^l_{i,j} \in \mathit{MatrixElements}$ позволяют корректно решить поставленную задачу для множеств путей длины l, где $l \in \{0, 1\}$. Таким образом, утверждение леммы справедливо для $k = 0$.

Индукционный переход: Предположим, что утверждение леммы справедливо для любого $k \leqslant (p-1)$, где $p \geqslant 1$, и покажем, что оно также справедливо для $k = p$.

Операции цикла в строках 9–11 алгоритма означают, что значение $T^{A,p}[i,j] = T^{A,p-1}[i,j] \bigoplus_{A \to BC \in P} (T^{B,p-1} \cdot T^{C,p-1})[i,j]$. Докажем, что значение в ячейке $T^{A,p}[i,j]$ позволяет корректно решить поставленную задачу для множества путей $\mathcal{P}_{\mathcal{G},p+1}(i,j,A)$.

Пусть значение $\bigoplus_{A \to BC \in P} (T^{B,p-1} \cdot T^{C,p-1})[i,j]$ позволяет корректно решить задачу для множества путей $\mathcal{P}'$. По индукционному предположению, значение в ячейке $T^{A,p-1}[i,j]$ позволяет это сделать для множества путей $\mathcal{P}_{\mathcal{G},p}(i,j,A)$. Тогда по условию леммы получаем, что значение $T^{A,p}[i,j] = T^{A,p-1}[i,j] \bigoplus_{A \to BC \in P} (T^{B,p-1} \cdot T^{C,p-1})[i,j]$ позволяет корректно решить поставленную задачу для множества путей $\mathcal{P}_{\mathcal{G},p}(i,j,A) \cup \mathcal{P}'$. Таким образом, для завершения доказательства леммы остаётся показать, что $\mathcal{P}_{\mathcal{G},p}(i,j,A) \cup \mathcal{P}' = \mathcal{P}_{\mathcal{G},p+1}(i,j,A)$.

Существование путей $\pi \in \mathcal{P}_{\mathcal{G},p+1}(i,j,A)$ равносильно существованию правила $A \to BC \in P$, а также путей $\pi_1 \in \mathcal{P}_{\mathcal{G},p}(i,r,B)$ и $\pi_2 \in \mathcal{P}_{\mathcal{G},p}(r,j,C)$, где путь π получается в результате конкатенации путей π_1 и π_2, а дерево вывода строки $\lambda(\pi)$ из нетерминала A с минимальной высотой $h \leqslant p+1$ представлено на рисунке 3.2. По индукционному предположению, значения $T^{B,p-1}[i,r]$ и $T^{C,p-1}[r,j]$ позволяют корректно решить поставленную задачу для множества всех таких путей π_1 и для множества всех таких путей π_2 соответственно. Используя свойства операции $\oplus$ и $\otimes$ из условия леммы, а также построение операций над матрицами получаем, что значение $(T^{B,p-1} \cdot T^{C,p-1})[i,j]$ позволяет корректно решить поставленную задачу для множества путей $\mathcal{P}_{BC}$, полученного конкатенацией таких путей π_1 и π_2. Кроме того, используя свойство операции $\oplus$ из условия леммы получаем, что $\bigoplus_{A \to BC \in P} (T^{B,p-1} \cdot T^{C,p-1})[i,j]$ позволяет корректно решить поставленную задачу для множества путей $\mathcal{P}' = \bigcup_{A \to BC \in P} \mathcal{P}_{BC}$. Таким образом, $\mathcal{P}_{\mathcal{G},p}(i,j,A) \cup \mathcal{P}' = \mathcal{P}_{\mathcal{G},p}(i,j,A) \cup \bigcup_{A \to BC \in P} \mathcal{P}_{BC} = \mathcal{P}_{\mathcal{G},p+1}(i,j,A)$, что доказывает утверждение леммы.

□

Следствием леммы 3.2.2 является следующая теорема о корректности алгоритма, представленного на листинге 4.

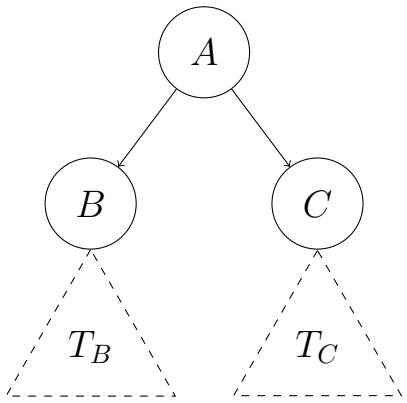


Рисунок 3.2 — Дерево вывода минимальной высоты $h = 1 + max(h_1, h_2)$ для строки $\lambda(\pi)$, где T_B и T_C — деревья вывода для строк $\lambda(p_1)$ и $\lambda(\pi_2)$ с высотами h_1 и h_2 соответственно

Теорема 3.2.3 (Корректность алгоритма)**.** Пусть $\mathcal{G} = \langle V, E, L\rangle$ — входной граф и $G = \langle N, \Sigma, P, S\rangle$ — входная КС-грамматика для алгоритма, представленного на листинге 4. Пусть выбранная алгебраическая структура удовлетворяет свойствам, представленным в условии леммы 3.2.2. Тогда для любой пары вершин (i, j) графа $\mathcal{G}$, для любого нетерминала $A \in N$, значение в ячейке $T^A[i, j]$ позволяет корректно решить поставленную задачу для множества всех путей графа $\mathcal{G}$ из вершины i в вершину j.

Доказательство. Алгоритм возвращает множество матриц T только в случае, когда для некоторого $k \geqslant 1$ $T^{A,k} = T^{A,k-1}$ для всех нетерминалов $A \in N$. То есть для любого нетерминала A $T^{A,p} = T^A$ для любого $p \geqslant k$. Таким образом, по лемме 3.2.2, для любой пары вершин (i, j) графа $\mathcal{G}$, для любого нетерминала $A \in N$, ячейка $T^A[i, j]$ позволяет корректно решить поставленную задачу для множества всех путей $i\pi j$ таких, что существует дерево вывода для строки $\lambda(\pi)$ из нетерминала A грамматики G. Что в свою очередь доказывает утверждение теоремы. □

Используя теорему 3.2.3 и индукцию на высоты деревьев вывода, соответствующих восстанавливаемым путям с помощью предложенного алгоритма для задач поиска одного пути и поиска всех путей, может быть доказана следующая теорема.

Теорема 3.2.4 (Корректность восстановления путей)**.** Пусть $\mathcal{G} = \langle V, E, L\rangle$ — входной граф, $G = \langle N, \Sigma, P, S\rangle$ — входная КС-грамматика и T — множество

матриц, возвращаемое алгоритмом, представленным на листинге 4. Пусть выбранная алгебраическая структура удовлетворяет свойствам, представленным в условии леммы 3.2.2. Тогда для любой пары вершин (i, j) и любого нетерминала $A \in N$:

- для задачи поиска одного пути алгоритм, представленный на листинге 5 построит путь $i\pi j$ такой, что существует дерево вывода для строки $\lambda(\pi)$ из нетерминала A грамматики G, если такой путь существует;
- для задачи поиска всех путей алгоритм, представленный на листинге 6 построит множество всех путей $i\pi j$ таких, что существует дерево вывода для строки $\lambda(\pi)$ из нетерминала A грамматики G.

Свойства предложенных алгебраических структур. Осталось показать, что предложенные в разделе 3.1 алгебраические структуры для задач достижимости, поиска одного и поиска всех путей обладают свойствами, обеспечивающими завершаемость и корректность предложенного алгоритма.

Во-первых, покажем завершаемость алгоритма для предложенных структур. Конечность алгебраических структур для задачи достидимости и для задачи поиска всех путей очевидна. Покажем, что предложенная структура для задачи поиска одного пути также конечна. Для этого остаётся показать, что для заданных графа и КС-грамматики максимальная высота h из минимальных высот деревьев вывода для различных троек (A, i, j) может быть оценена сверху, а именно: $h \leqslant |N||V|^2$. Каждый внутренний узел рассматриваемых в предложенном алгоритме деревьев ассоциируется с некоторым нетерминалом A и поддеревом с корнем в этом узле. Листья такого поддерева образуют строку, соответствующую некоторому пути в графе из вершины i в вершину j. Поэтому с каждым внутренним узлом таких деревьев будем ассоциировать тройку (A, i, j). Если бы для некоторой тройки (A, i, j) существовало дерево вывода минимальной высоты $h > |N||V|^2$, то в таком дереве существовал бы путь длины $h > |N||V|^2$ из корня дерева до некоторого листа. А так как существует лишь $|N||V|^2$ различных таких троек, то на этом пути найдутся два различных узла u и v, которые ассоциируются с одной и той же тройкой (B, k, l). Пусть узел u расположен ближе к корню дерева, чем узел v. Тогда заменив в этом дереве поддерево, соответствующее узлу u, на поддерево, соответствующее узлу v, мы получим новое дерево вывода для тройки (A, i, j), в котором уменьшились некоторые расстояния от корня дерева до листов. Такие преоб-

разования можно повторять, пока высота дерева не станет меньше либо равна $|N||V|^2$. Таким образом, изначальное дерево не обладало минимальной высотой для тройки (A, i, j). Значит все такие высоты $h \leqslant |N||V|^2$, а четвёрки, которые будут использоваться в предложенном алгоритме, содержат $height \leqslant |N||V|^2+1$. Поэтому предложенная структура для задачи поиска одного пути также имеет конечный носитель.

Для завершаемости алгоритма с использованием предложенных структур остается показать, что на элементах этих структур может быть задано отношение частичного порядка $\preceq$ такое, что для любой пары вершин (i, j) и любого $k \geqslant 1$, $T^{A,k-1}[i, j] \preceq T^{A,k}[i, j]$.

Пусть $\preceq_{rel}$ — отношение частичного порядка соответствующее задаче достижимости, $\preceq_{single}$ — задаче поиска одного пути и $\preceq_{all}$ — задаче поиска всех путей в графе. Для краткости будем использовать обозначения z_l, z_r, z_h, z_m и z_{ms} для *z.left*, *z.right*, *z.height*, *z.middle* и *z.middles* соответственно, где z — элемент носителя *MatrixElements* одной из трёх введенных алгебраических структур. Тогда для любых элементов x и y этих носителей определим отношения частичного порядка следующим образом:

$$
\begin{aligned}
x \preceq_{rel} y \quad &\Longleftrightarrow \quad (x = 0) \wedge (y = 1), \\
x \preceq_{single} y \quad &\Longleftrightarrow \quad (x = \bot = (0, 0, 0, 0)) \vee \\
&\qquad\quad ((x_l = y_l) \wedge (x_r = y_r) \wedge ((y_h, y_m) \leqslant (x_h, x_m))), \\
x \preceq_{all} y \quad &\Longleftrightarrow \quad (x = \bot = (0, 0, \varnothing)) \vee ((x_l = y_l) \wedge (x_r = y_r) \wedge (x_{ms} \subseteq y_{ms})).
\end{aligned}
$$

Необходимо показать, что для любого $k \geqslant 1$, $T^{A,k-1}[i, j] \preceq_{rel} T^{A,k}[i, j]$ для задачи достижимости, $T^{A,k-1}[i, j] \preceq_{single} T^{A,k}[i, j]$ для задачи поиска одного пути и $T^{A,k-1}[i, j] \preceq_{all} T^{A,k}[i, j]$ для задачи поиска всех путей в графе. На очередной итерации цикла в строках 9–11 алгоритма, значение в ячейке $T^A[i, j]$ либо может не изменится, либо изменится в результате выполнения операций $T^A \leftarrow T^A \bigoplus (T^B \cdot T^C)$ в строке 11. Если значение не изменилось, то утверждение теоремы очевидно в силу рефлексивности рассматриваемых отношений частичного порядка. В противном случае $T^{A,k}[i, j] = T^{A,k-1}[i, j] \oplus (T^{B,k-1} \cdot T^{C,k-1})[i, j]$. Для всех трёх задач поиска путей в графе с заданными КС-ограничениями докажем, что $T^{A,k-1}[i, j] \preceq T^{A,k-1}[i, j] \oplus (T^{B,k-1} \cdot T^{C,k-1})[i, j]$ для соответствующего отношения частичного порядка $\preceq$.

Задача достижимости. Используя алгебраическую структуру над логическими значениями и операцию дизъюнкции в качестве операции $\oplus$, значение

в ячейке $T^A[i,j]$ может изменится только с 0 на 1. Поэтому $T^{A,k-1}[i,j] \preceq_{rel} T^{A,k}[i,j]$, что доказывает завершаемость алгоритма для задачи достижимости.

Задача поиска одного пути. По определению операции $\oplus$ полукольца над множеством четвёрок *PathIndex*, значение в ячейке $T^A[i,j]$ может изменится только если $T^{A,k-1}[i,j] = \perp$ или

$$((T^{A,k}[i,j].height, T^{A,k}[i,j].middle) \leqslant (T^{A,k-1}[i,j].height, T^{A,k-1}[i,j].middle)).$$

Если $T^{A,k-1}[i,j] = \perp$, то по определению отношения частичного порядка $\preceq_{single}$, $T^{A,k-1}[i,j] \preceq_{single} T^{A,k}[i,j]$. Иначе по определениям операции умножения матриц $(\cdot)$ и операции умножения элементов матриц $\otimes$, получаем, что $T^{A,k-1}[i,j].left = (T^{B,k-1} \cdot T^{C,k-1})[i,j].left$ и $T^{A,k-1}[i,j].right = (T^{B,k-1} \cdot T^{C,k-1})[i,j].right$. Таким образом, по определению отношения частичного порядка $\preceq_{single}$, $T^{A,k-1}[i,j] \preceq_{single} T^{A,k}[i,j]$, что доказывает завершаемость алгоритма для задачи поиска одного пути.

Задача поиска всех путей. По определению операции $\oplus$ полукольца над множеством троек *AllPathIndex*, значение в ячейке $T^A[i,j]$ может изменится только если $T^{A,k-1}[i,j] = \perp$ или $(T^{A,k-1}[i,j].middles \subseteq T^{A,k}[i,j].middles)$. Если $T^{A,k-1}[i,j] = \perp$, то утверждение теоремы доказано по определению отношения частичного порядка $\preceq_{all}$. Иначе по определениям операции умножения матриц $(\cdot)$ и операции умножения элементов матриц $\otimes$, получаем, что $T^{A,k-1}[i,j].left = (T^{B,k-1} \cdot T^{C,k-1})[i,j].left$ и $T^{A,k-1}[i,j].right = (T^{B,k-1} \cdot T^{C,k-1})[i,j].right$. Таким образом, по определению отношения частичного порядка $\preceq_{all}$, $T^{A,k-1}[i,j] \preceq_{all} T^{A,k}[i,j]$, что доказывает завершаемость алгоритма для задачи поиска всех путей.

Во-вторых, докажем корректность алгоритма для предложенных алгебраических структур, показав, что они удовлетворяют следующим свойствам, указанным в условии леммы 3.2.2.

Все три предложенные алгебраические структуры являются полукольцами без требования ассоциативности операции умножения. Осталось показать справедливость свойств для элементов $\alpha^l_{i,j}$ и для операций сложения и умножения предложенных структур.

Задача достижимости. В предложенной алгебраической структуре для задачи достижимости были выделены элементы $\alpha^0_{i,j} = \alpha^1_{i,j} = 1$, которые используются в алгоритме, представленном на листинге 4, в случае существования пути из $\mathcal{P}_{\mathcal{G},1}(i,j,A)$ для некоторого нетерминала $A \in N$. Если такого пути не

существует, то в соответствующей ячейке будет значение $\perp = 0$. Таким образом, элементы $\alpha^0_{i,j}$ и $\alpha^1_{i,j}$ позволяют определить наличие путей длины 0 или 1, удовлетворяющих заданным КС-ограничениям, что доказывает соответствующее свойство алгебраической структуры $\langle\{0,1\},\vee,\wedge,0\rangle$. Справедливость свойства для операции $\oplus$ очевидна, так как $\alpha_1 \vee \alpha_2 = 1$ только если существует хотя бы один искомый путь в множестве $\mathcal{P}_1$ или в множестве $\mathcal{P}_2$, что эквивалентно решению задачи для множества $\mathcal{P}_1 \cup \mathcal{P}_2$. И, наконец, если элементы α_1 и α_2 позволяют корректно решить задачу достижимости для множества путей $\mathcal{P}_1$ из вершины i в вершину k и для множества $\mathcal{P}_2$ путей из вершины k в вершину j, то элемент $\alpha_1 \wedge \alpha_2 = 1$, только если существовал хотя бы один искомый путь в множестве $\mathcal{P}_1$ и хотя бы одни искомый путь в множестве $\mathcal{P}_2$. Конкатенация таких путей является искомым путём из вершины i в вершину j, поэтому значение $\alpha_1 \wedge \alpha_2$ позволит решить задачу для множества путей $\mathcal{P}_1 \cdot \mathcal{P}_2$.

Задача поиска одного пути. Рассмотрим предложенную структуру $\langle \mathit{PathIndex}, \oplus, \otimes, \perp\rangle$ для задачи поиска одного пути. Для корректного решения поставленной задачи будем использовать элемент $\perp = (0,0,0,0) \in \mathit{PathIndex}$ для индикации отсутствия искомых путей и элемент $(\mathit{left}, \mathit{right}, \mathit{middle}, \mathit{height}) \in \mathit{PathIndex}$ если существует искомый путь π из вершины *left* в вершину *right*, с некоторой выделенной вершиной *middle* и *height* — минимальной высотой дерева вывода строки $\lambda(\pi)$ для соответствующего нетерминала грамматики. Остальные свойства следуют из построения этой алгебраической структуры в разделе 3.1:

- выделенные элементы $\alpha^0_{i,i} = (i,i,i,1)$ и $\alpha^1_{i,j} = (i,j,i,1)$ описывают искомые пути длины 0 или 1 с высотой дерева вывода 1, если такие пути существуют;
- если элементы α_1 и α_2 описывают искомые пути с минимальной высотой дерева вывода для множеств путей $\mathcal{P}_1$ и $\mathcal{P}_2$ из вершины i в вершину j, то элемент $\alpha_1 \oplus \alpha_2$ описывает искомый путь с минимальной высотой дерева вывода для множества путей $\mathcal{P}_1 \cup \mathcal{P}_2$;
- если элементы α_1 и α_2 описывают искомые пути с минимальной высотой дерева вывода для множества путей $\mathcal{P}_1$ из вершины i в вершину k и для множества $\mathcal{P}_2$ путей из вершины k в вершину j, то элемент $\alpha_1 \otimes \alpha_2$ описывает искомый путь с минимальной высотой дерева вывода для множества путей $\mathcal{P}_1 \cdot \mathcal{P}_2$.

Задача поиска всех путей. Для задачи поиска всех путей с заданными КС-ограничениями в предложенной структуре $\langle AllPathIndex, \oplus, \otimes, \perp \rangle$ используется элемент $\perp = (0, 0, \varnothing) \in AllPathIndex$ для индикации отсутствия искомых путей и элемент $(left, right, middles) \in AllPathIndex$ если существуют искомые пути π из вершины $left$ в вершину $right$, с некоторой выделенной вершиной $middle \in middles$. Остальные свойства следуют из построения этой алгебраической структуры в разделе 3.1:

- с использованием специального значения $n = |V|$ в множестве $middles$ выделенные элементы $\alpha^0_{i,j} = \alpha^1_{i,j} = (i, j, \{n\})$ описывают все искомые пути длины 0 или 1;
- если элементы α_1 и α_2 описывают все искомые пути для множеств путей $\mathcal{P}_1$ и $\mathcal{P}_2$ из вершины i в вершину j, то элемент $\alpha_1 \oplus \alpha_2$ описывает все искомые пути для множества путей $\mathcal{P}_1 \cup \mathcal{P}_2$;
- если элементы α_1 и α_2 описывают все искомые пути для множества путей $\mathcal{P}_1$ из вершины i в вершину k и для множества $\mathcal{P}_2$ путей из вершины k в вершину j, то элемент $\alpha_1 \otimes \alpha_2$ описывает все искомые пути для множества путей $\mathcal{P}_1 \cdot \mathcal{P}_2$.

Таким образом, для всех трёх предложенных в разделе 3.1 алгебраических структур для задач достижимости, поиска одного и поиска всех путей в графе с заданными КС-ограничениями справедливы теоремы 3.2.1 и 3.2.3.

3.3 Временная сложность алгоритма

В данном разделе представлены оценки временной сложности предложенного алгоритма для рассмотренных задач поиска путей в графе.

Пусть для алгоритма, представленного на листинге 4 была выбрана алгебраическая структура $\langle MatrixElements, \oplus, \otimes, \perp \rangle$ с конечным носителем $MatrixElements$ и с заданным на нём отношением частичного порядка $\preceq$ таким, что для любой пары вершин (i, j) и любого $k \geqslant 1$, $T^{A,k-1}[i, j] \preceq T^{A,k}[i, j]$. Тогда по теореме 3.2.1 алгоритм завершит свою работу за конечное число шагов. Пусть N_{iter} — максимальное количество итераций цикла в строках 9–11 алгоритма, представленного на листинге 4. Стоит отметить, что значение N_{iter} зависит от выбранной алгебраической структуры.

Также предположим, что все операции над элементами из множества *MatrixElements* вычисляются за $O(1)$ элементарных операций. Кроме того, для использованной алгебраической структуры оценим количество элементарных операций, необходимых в худшем случае для вычисления операций над матрицами размера $n \times n$, как:

- $O(n^2)$ — для операции $\bigoplus$ поэлементного сложения двух матриц,
- $O(n^3)$ — для операции $(\cdot)$ умножения двух матриц.

Стоит отметить, что данные оценки могут быть улучшены для алгебраических структур, обладающих некоторыми свойствами. Например, существует алгоритм с субкубической сложностью $O(n^{3-\varepsilon})$, где $\varepsilon > 0$, для таких алгебраических структур, как кольца [74].

С учётом сделанных предположений справедлива следующая оценка временной сложности алгоритма поиска путей в графе с заданными КС-ограничениями.

Теорема 3.3.1 (Оценка временной сложности алгоритма поиска путей)**.** Пусть $\mathcal{G} = \langle V, E, L\rangle$ — входной граф, $G = \langle N, \Sigma, P, S\rangle$ — входная КС-грамматика. Тогда для алгоритма, представленного на листинге 4, справедлива следующая оценка временной сложности: $O(|N||P||V|^5)$.

Доказательство. На этапе инициализации матриц алгоритм производит запись значений в ячейки этих матриц. В строке 5 производится операция записи $O(|E|)$ раз, а в строке 8 — $O(|V||N|)$ раз. Поэтому для всех задач поиска путей в графе на этапе инициализации производится $O(|E| + |V||N|)$ элементарных операций.

Далее рассмотрим цикл в строках 9–11, вычисляющий транзитивное замыкание. Алгоритм продолжает вычисления пока любая матрица из множества T меняется. Всего матриц в этом множестве $|N|$ и каждая имеет $|V|^2$ элементов. Из-за свойств трёх предложенных алгебраических структур значения ячеек матриц монотонно возрастают относительно соответствующего отношения частичного порядка. Далее рассмотрим максимальное количество итераций N_{iter}, которое может возникнуть при решении различных задач поиска путей в графе.

Задача достижимости и задача поиска одного пути. С каждой итерацией цикла в строках 9–11 увеличивается на единицу рассматриваемые высоты деревьев вывода строк, соответствующих путям в графе. Для задач достижимости и поиска одного пути значение конкретной ячейки может изменится лишь

в рамках одной итерации цикла в строках 9–11. Поэтому максимальное количество итераций рассматриваемого цикла может быть достигнуто в случае, когда за каждую итерацию цикла изменяется значение лишь одной ячейки в одной матрице. Для задачи достижимости значение может изменится с $\perp = 0$ на 1, а для задачи поиска одного пути значение либо изменяется с $\perp = (0, 0, 0, 0)$ на четвёрку, описывающую некоторый найденный на данной итерации путь, либо значение для найденного пути изменяется на значение для другого пути с меньшим номером промежуточной вершины *middle*. В рамках одной итерации цикла в строках 9–11 операции в строке 11 выполняются $O(|P|)$ раз. Поэтому для задач достижимости и поиска одного пути справедлива следующая оценка временной сложности: $O(N_{iter}|P||V|^3)$, где максимальное количество итераций цикла в строках 9–11 $N_{iter} = |N||V|^2$.

Задача поиска всех путей. Для данной задачи докажем, что количество итераций цикла в строках 9–11 превышает максимальное количество таких итераций при решении задачи поиска одного пути не более чем на 1, то есть всегда меньше либо равно, чем $|N||V|^2 + 1$. Для этого покажем, что изменения значения в одной из матриц на итерации с номером k для задачи поиска всех путей влечёт за собой тот факт, что при решении с такими же входными данными задачи поиска одного пути значение в одной из матриц изменится на итерации с номером k или $k - 1$.

Аналогично уже рассмотренным задачам, для задачи поиска всех путей с каждой итерацией цикла в строках 9–11 увеличивается на единицу рассматриваемые высоты деревьев вывода строк, соответствующих путям в графе. Пусть на итерации с номером k изменилось значение в ячейке $T^A[i, j]$ после обнаружения путей из вершины i в вершину j, образующих строки, выводимые из нетерминала A и имеющие деревья вывода высоты $k + 1$. Это могло произойти в двух случаях: либо $T^{A,k-1}[i, j] = \perp$ и $k + 1$ — минимальная высота деревьев вывода для таких путей, либо $T^{A,k-1}[i, j] = (i, j, middles) \neq \perp$ и был найден хотя бы один путь π с выделенной промежуточной вершиной $m \notin middles$. В первом случае, на такой же итерации будет изменено значение в ячейке $T^A[i, j]$ при решении задачи поиска одного пути, поэтому $k \leqslant |N||V|^2$. Больший интерес представляет собой второй случай, который рассмотрим подробнее. Так как $m \notin middles$, то $k+1$ — минимальная высота деревьев вывода для путей из вершины i в вершину j с выделенной промежуточной вершиной m, образующих строки, выводимые из нетерминала A. Пусть такое дерево вывода минималь-

ной высоты образовано с использованием на корневом уровне правила вывода $(A \to B\ C) \in P$, где путь $i\pi j$ — конкатенация путей $i\pi_1 m$ и $m\pi_2 j$, а $B \Rightarrow_G \lambda(\pi_1)$ и $C \Rightarrow_G \lambda(\pi_2)$. Тогда хотя бы одна из строк $\lambda(\pi_1)$ или $\lambda(\pi_2)$ имеет минимальную высоту дерева вывода из соответствующего нетерминала равную k. Иначе можно было бы построить дерево вывода строки $\lambda(\pi)$ с высотой меньшей, чем $k+1$. Таким образом, на предыдущей итерации цикла с номером $k-1$ был найден хотя бы один путь с минимальной высотой дерева вывода из соответствующего нетерминала и изменилось значение в ячейке $T^B[i,m]$ или в ячейке $T^C[m,j]$. Значения в этих ячейках также изменятся на $k-1$ итерации цикла при решении задачи поиска одного пути. Таким образом, для решения задачи поиска одного пути потребуется не менее $k-1$ итерации, а из уже доказанной оценки сверху на количество таких итераций получаем, что $k-1 \leqslant |N||V|^2$ и, значит, $k \leqslant |N||V|^2 + 1$.

Мы показали, что при решении задачи поиска всех путей количество итераций в цикле не превышает $|N||V|^2+1$ и поэтому асимптотически для задачи поиска всех путей справедлива оценка, полученная для предыдущих двух задач, а именно: $O(|N||P||V|^5)$. □

Следует отметить что приведённая оценка временной сложности алгоритма, представленного на листинге 4, может быть существенно улучшена для некоторых алгебраических структур или для разреженных графов. Однако в общем случае полученные оценки временной сложности алгоритма являются точными, так как существует пример графа и КС-грамматики, при которых эти оценки временной сложности достигаются. Впервые такой пример опубликовал Джелле Хеллингс в работе [46]. В этом примере используется граф, схожий с графом $\mathcal{G}_1$, изображённым на рис. 1.1, однако имеющий 2^n+1 дуг с меткой a и 2^n дуг с меткой b. А используемая КС-грамматика порождает язык $\mathcal{L} = \{a^n b^n \mid n \geqslant 1\}$. Взаимная простота длин двух циклов в этом графе приводит к тому, что минимальная высота дерева вывода путей из вершины 0 в вершину 0, удовлетворяющих заданным КС-ограничениям — $O(|V|^2)$. Пример работы предложенного алгоритма на частном случае таких графов будет рассмотрен в следующем разделе.

Далее, используя индукцию на минимальные высоты деревьев вывода, соответствующих восстанавливаемому пути с помощью алгоритма, представленного на листинге 5, может быть доказана следующая теорема.

Теорема 3.3.2 (Оценка временной сложности алгоритма восстановления одного пути)**.** Пусть $\mathcal{G} = \langle V, E, L \rangle$ — входной граф, $G = \langle N, \Sigma, P, S \rangle$ — входная КС-грамматика и T — множество матриц, возвращаемое алгоритмом, представленным на листинге 4, с использованием алгебраической структуры, соответствующего задаче поиска одного пути. Тогда для алгоритма, представленного на листинге 5, справедлива следующая оценка временной сложности: $O(d|N|^2)$, где d — количество вершин в дереве вывода минимальной высоты для восстановленного алгоритмом пути.

В данной работе временная сложность алгоритма, представленного на листинге 6, не приводится, так как она зависит от конкретного способа ограничения количества восстанавливаемых путей и построения соответствующих множеств искомых путей.

3.4 Пример

В данном разделе работа изложенного алгоритма продемонстрирована на примере, основанном на КС-языке правильных вложенных скобочных последовательностей.

Пусть даны граф $\mathcal{G}_1$, изображённый на рис. 1.1 и КС-ограничения в виде КС-языка $\mathcal{L} = \{a^n b^n \mid n \geqslant 1\}$. Представим данный КС-язык в виде КС-грамматики G в ослабленной нормальной форме Хомского со следующими правилами вывода.

$$\begin{array}{llll} 0: & S \to A\ B & 3: & A \to a \\ 1: & S \to A\ S_1 & 4: & B \to b \\ 2: & S_1 \to S\ B & & \end{array}$$

Далее разберём работу алгоритма, представленного на листинге 4, для задачи достижимости, поиска одного и поиска всех путей.

Задача достижимости. Для рассматриваемых графа и КС-грамматики алгоритм, представленный на листинге 4, инициализирует множество матриц T, изображенных на рис. 3.3.

Так как в заданной грамматике отсутствуют правила вида $A \to \varepsilon$, то алгоритм переходит к выполнению цикла в строках 9–11. Далее в процессе

$$T^{A,0} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$T^{B,0} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Рисунок 3.3 — Множество матриц T после инициализации для задачи достижимости (элементы $T^{S_1,0}[i,j]$ и $T^{S,0}[i,j]$ равны $\perp = 0$ для всех i,j)

работы алгоритма будут меняться только матрицы T^S и T^{S_1}. После первой итерации цикла все элементы матрицы $T^{S_1,1}$ останутся равны 0, а элементы матрицы $T^{S,1}$ изображены на рис. 3.4. При нахождении очередного пути на некоторой итерации алгоритм добавляет информацию о достижимости в соответствующую матрицу. Например, на первой итерации цикла был найден путь $2\pi 3 = (2,a,0),(0,b,3)$, образующий строку $\lambda(\pi) = ab$, выводимую из нетерминала S. Информация о достижимости вершины 3 из вершины 2 добавляется в ячейку $T^{S,1}[2,3]$.

$$T^{S,1} = T^{S,0} \bigoplus (T^{A,0} \cdot T^{B,0}) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Рисунок 3.4 — Элементы матрицы $T^{S,1}$, полученные после первой итерации для задачи достижимости

Для полного обхода графа в соответствии с входной КС-грамматикой этот процесс повторяется, пока добавляется новая информация хотя бы в одну из матриц. В данном примере алгоритм выполнит 13 итераций цикла, после чего завершит свою работу. Результирующая матрица $T^{S,13}$ представлена на рис. 3.5.

Наконец, после работы алгоритма можно использовать получившуюся матрицу для построения множества пар вершин, являющегося ответом на зада-

$$T^{S,13} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Рисунок 3.5 — Элементы результирующей матрицы $T^{S,13}$ для задачи достижимости

чу достижимости в графе с заданными КС-ограничениями. В данном примере этим множеством является $\{(0,0),(0,3),(1,0),(1,3),(2,0),(2,3)\}$.

Задача поиска одного пути. Аналогично задаче достижимости, продемонстрируем работу алгоритма, представленного на листинге 4, для задачи поиска одного пути, при которой используется соответствующее полукольцо $\langle PathIndex, \oplus, \otimes, \bot \rangle$, где $\bot = (0,0,0,0)$. Алгоритм инициализирует множество матриц T, изображенных на рис. 3.6.

$$T^{A,0} = \begin{pmatrix} \bot & (0,1,0,1) & \bot & \bot \\ \bot & \bot & (1,2,1,1) & \bot \\ (2,0,2,1) & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

$$T^{B,0} = \begin{pmatrix} \bot & \bot & \bot & (0,3,0,1) \\ \bot & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \\ (3,0,3,1) & \bot & \bot & \bot \end{pmatrix}$$

Рисунок 3.6 — Множество матриц T после инициализации для задачи поиска одного пути (элементы $T^{S_1,0}[i,j]$ и $T^{S,0}[i,j]$ равны $\bot = (0,0,0,0)$ для всех i,j)

Как и в случае решения задачи достижимости, далее в процессе работы алгоритма будут меняться только матрицы T^S и T^{S_1}. После первой итерации цикла в строках 9–11 все элементы матрицы $T^{S_1,1}$ останутся равны $\bot$, а элементы матрицы $T^{S,1}$ изображены на рис. 3.7. В отличие от предыдущего случая, при решении задачи поиска одного пути в матрицу добавляется информация не только о достижимости, но и о конечных вершинах пути, одной из его промежуточных вершин, а также о высоте соответствующего дерева вывода. Например,

на первой итерации цикла был найден путь $2\pi 3 = (2, a, 0), (0, b, 3)$, образующий строку $\lambda(\pi) = ab$, выводимую из нетерминала S. В ячейку $T^{S,1}[2, 3]$ записывается значение $(2, 3, 0, 2)$, так как был найден путь из вершины 2 в вершину 3, с промежуточной вершиной 0 и с деревом вывода строки $\lambda(\pi) = ab$ высоты 2, изображенного на рис. 3.8.

$$T^{S,1} = T^{S,0} \bigoplus (T^{A,0} \cdot T^{B,0}) = \begin{pmatrix} \bot & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \\ \bot & \bot & \bot & (2, 3, 0, 2) \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

Рисунок 3.7 — Элементы матрицы $T^{S,1}$, полученные после первой итерации для задачи поиска одного пути

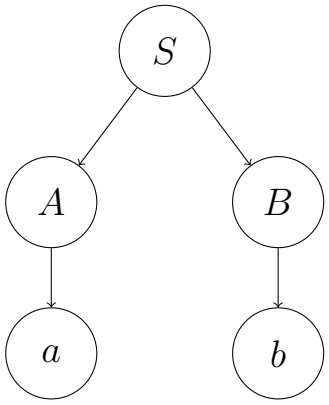


Рисунок 3.8 — Дерево вывода из нетерминала S для строки $\lambda(\pi) = ab$ минимальной высоты $h = 2$

Как и при решении задачи достижимости алгоритм выполнит 13 итераций цикла в строках 9–11, после чего завершит свою работу. Результирующая матрица $T^{S,13}$ для задачи поиска одного пути представлена на рис. 3.9.

$$T^{S,13} = \begin{pmatrix} (0, 0, 1, 12) & \bot & \bot & (0, 3, 1, 6) \\ (1, 0, 2, 4) & \bot & \bot & (1, 3, 2, 10) \\ (2, 0, 0, 8) & \bot & \bot & (2, 3, 0, 2) \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

Рисунок 3.9 — Элементы результирующей матрицы $T^{S,13}$ для задачи поиска одного пути

Построенные матрицы также содержат информацию о достижимости, поэтому аналогично может быть построено множество пар вершин $\{(0,0),(0,3),(1,0),(1,3),(2,0),(2,3)\}$, которое является ответом на задачу достижимости с заданными КС-ограничениями. Однако построенные матрицы также содержат информацию, достаточную для построения по одному пути, соответствующему заданным ограничениям, для каждой пары вершин из этого множества. Для этого может быть использован алгоритм, представленный на листинге 5. Например, для восстановления такого пути из вершины 0 в вершину 0 необходимо передать этому алгоритму на вход: начальную и конечную вершины $i = j = 0$; S — стартовый нетерминал грамматики; результирующие матрицы для всех нетерминалов, вычисленные алгоритмом, представленным на листинге 4; а также КС-ограничения в виде грамматики G. Тогда алгоритм, представленный на листинге 5, вернёт путь $0\pi0$, для которого $\lambda(\pi) = a^6b^6$. Минимальная высота дерева вывода такой строки из стартового нетерминала S равна 12, что соответствует значению $(0,0,1,12)$, записанному в ячейке $T^{S,13}[0,0]$.

Задача поиска всех путей. Аналогично предыдущим двум задачам, предоставим состояния матриц из множества T для задачи поиска всех путей и полукольца $\langle AllPathIndex, \oplus, \otimes, \bot\rangle$, где $\bot = (0,0,\varnothing)$. Множество матриц T после инициализации изображено на рис. 3.10 (в процессе инициализации матриц вместо промежуточных вершин используется специальное значение $4 = |V|$, сигнализирующее о том, что длина найденного пути равна 0 или 1). Элементы матрицы $T^{S,1}$, полученные после первой итерации цикла в строках 9–11 изображены на рис. 3.11. И, наконец, результирующая матрица $T^{S,13}$ для задачи поиска всех путей представлена на рис. 3.12.

Можем заметить, что для данного примера в результирующей матрице T^S нету элементов с более чем одной промежуточной вершиной. Поэтому полученная матрица очень схожа с соответствующей матрицей для задачи поиска одного пути, но без наличия информации о высотах деревьев вывода. Однако в общем случае количество найденных промежуточных вершин для путей между двумя вершинами может быть больше одного. После построения алгоритмом таких матриц известно, что они содержат все такие промежуточные вершины, поэтому с помощью алгоритма, представленного на листинге 6, могут быть

$$T^{A,0} = \begin{pmatrix} \bot & (0,1,\{4\}) & \bot & \bot \\ \bot & \bot & (1,2,\{4\}) & \bot \\ (2,0,\{4\}) & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

$$T^{B,0} = \begin{pmatrix} \bot & \bot & \bot & (0,3,\{4\}) \\ \bot & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \\ (3,0,\{4\}) & \bot & \bot & \bot \end{pmatrix}$$

Рисунок 3.10 — Множество матриц T после инициализации для задачи поиска всех путей (элементы $T^{S_1,0}[i,j]$ и $T^{S,0}[i,j]$ равны $\bot = (0,0,\varnothing)$ для всех i,j)

$$T^{S,1} = \begin{pmatrix} \bot & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \\ \bot & \bot & \bot & (2,3,\{0\}) \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

Рисунок 3.11 — Элементы матрицы $T^{S,1}$, полученные после первой итерации для задачи поиска всех путей

$$T^{S,13} = \begin{pmatrix} (0,0,\{1\}) & \bot & \bot & (0,3,\{1\}) \\ (1,0,\{2\}) & \bot & \bot & (1,3,\{2\}) \\ (2,0,\{0\}) & \bot & \bot & (2,3,\{0\}) \\ \bot & \bot & \bot & \bot \end{pmatrix}$$

Рисунок 3.12 — Элементы результирующей матрицы $T^{S,13}$ для задачи поиска всех путей

восстановлены все пути, удовлетворяющие заданным КС-ограничениям. Например, могут быть восстановлены все пути из вершины 0 в вершину 0, образующие строки из языка $\mathcal{L} = \{a^n b^n \mid n \geqslant 1\}$. Множество таких путей бесконечно и они образуют слова из множества $\{a^6b^6, a^{12}b^{12}, a^{18}b^{18}, \ldots\}$. Для любого $m \geqslant 1$, путь, образующий строку $a^{6m}b^{6m}$, проходит по циклу с метками a $2m$ раз и по циклу с метками b — $3m$ раз. При реализации алгоритма восстановления всех путей необходимо ограничивать количество восстанавливаемых путей и добавить соответствующий входной параметр в алгоритм.

Таким образом, была продемонстрирована работа изложенного алгоритма поиска путей в графе с заданными КС-ограничениями.

3.5 Реализация

В данном разделе приведены детали реализации полученного алгоритма поиска путей в графе с заданными КС-ограничениями с использованием операций умножения матриц. Все перечисленные в разделе реализации находятся в открытом доступе в рамках платформы *CFPQ_PyAlgo*[1].

При реализации предложенного алгоритма учитывались особенности реальных графов в таких областях, как анализ RDF данных и статический анализ программ. Такие графы имеют довольно большие размеры (до десятков миллионов вершин), однако являются разреженными (имеют до десятков миллионов дуг). Для графов таких размеров целесообразно использовать параллельные вычислительные системы, так как время, затраченное на организацию параллельных вычислений (например, обмен данными между CPU и GPU) будет незначителен в сравнении с временем самого анализа. Поэтому обязательными атрибутами реализаций являются: использование параллельных вычислений и представление матриц с помощью разреженных форматов. Так как рассматриваемые графы не являются сильно разреженными, то для хранения разреженных матриц был выбран формат CSR. Далее будут описаны полученные реализации предложенного алгоритма.

Задача достижимости. Для решения задачи достижимости были использованы операции умножения и сложения над набором булевых матриц. А для реализации данного алгоритма были выбраны библиотеки для параллельного вычисления этих операций над разреженными булевыми матрицами на CPU и GPU. Реализация на CPU была создана с использованием библиотеки SuiteSparse:GraphBLAS, а для реализации на GPU — с использованием библиотеки cuBool. Стоит отметить, что библиотека SuiteSparse:GraphBLAS содержит

[1]CFPQ_PyAlgo — платформа для разработки и тестирования алгоритмов поиска путей в графе с заданными КС-ограничениями: https://github.com/JetBrains-Research/CFPQ_PyAlgo (дата обращения: 14.01.2022).

встроенные алгебраические структуры, которые могут быть использованы для вычисления операций над булевыми матрицами. В результате были получены следующие реализации:

- $MtxReach_{CPU}$ — на языке `Python` с использованием пакета pygraphblas, являющегося обёрткой для библиотеки SuiteSparse:GraphBLAS;
- $MtxReach_{GPU}$ — на языке `Python` с использованием пакета pycubool, являющегося обёрткой для библиотеки cuBool.

Для решения остальных задач поиска путей в графе с заданными КС-ограничениями были использованы более сложные типы для элементов матриц. Это приводит к существенному увеличению затрат по памяти, что особенно критично для реализаций на GPU. Предложенный алгоритм, основанный на умножении матриц, для задач поиска одного и всех путей в графе может быть реализован на GPU, например, с использованием библиотеки CUSP, которая позволяет вычислять операции над матрицами с пользовательским типом данных. Однако для апробации таких реализаций на реальных данных необходимо проведение дальнейших исследований по оптимизации предложенного алгоритма и использованных алгебраических структур для задач поиска одного и всех путей в графе. Особое внимание в таком исследовании необходимо уделить затратам по памяти предложенного алгоритма и деталям реализации вычисления на GPU операций над матрицами в конкретной выбранной библиотеке. В данной работе предложенный алгоритм для задач поиска одного и всех путей был реализован только на CPU.

Задача поиска одного пути. Алгоритм для решения задачи поиска одного пути был также реализован с использованием библиотеки SuiteSparse:GraphBLAS. Однако вместо использования одной из встроенных алгебраических структур, для вычисления операций над матрицами в этом алгоритме необходимо определить пользовательский тип данных и операции над ними, соответствующие приведённой для этой задачи структуре $\langle PathIndex, \oplus, \otimes, \perp \rangle$. В результате была получены следующая реализация:

- $MtxSingle_{CPU}$ — на языке `Python` с использованием пакета pygraphblas, являющегося обёрткой для библиотеки SuiteSparse:GraphBLAS.

Задача поиска всех путей. Реализация для задачи поиска всех путей была также получена с использованием библиотеки SuiteSparse:GraphBLAS. Однако

пакет pygraphblas на момент написания данного текста не позволяет определить пользовательский тип данных и операции над ними, соответствующие алгебраической структуре $\langle AllPathIndex, \oplus, \otimes, \perp \rangle$. Поэтому реализация алгоритма для поиска всех путей была написана на языке `C++` с использованием библиотеки SuiteSparse:GraphBLAS, а также для этого алгоритма была написана собственная обёртка на языке `Python`. В итоге была получена следующая реализация:

- $MtxAll_{CPU}$ — на языке `Python` с использованием собственной обёртки для реализации алгоритма на языке `C++`, использующей библиотеку SuiteSparse:GraphBLAS.

Кроме того, для задач поиска одного и поиска всех путей в графе были реализованы на языке `Python` алгоритмы восстановления искомых путей по информации, содержащейся в построенных матрицах. Стоит отметить, что в реализации алгоритма восстановления всех путей был дополнительно использован параметр, ограничивающий длины восстанавливаемых путей для завершаемости этого процесса.

Глава 4. Алгоритм поиска путей в графе с заданными КС-ограничениями не требующий трансформации КС-грамматики

В данной главе изложен алгоритм поиска путей в графе с заданными КС-ограничениями, который не требуют трансформации входной КС-грамматики, а также использует операции линейной алгебры для задач достижимости, поиска одного пути и поиска всех путей. Также сформулированы и доказаны утверждения о корректности и временной сложности полученного алгоритма. Кроме того, приведены детали реализаций алгоритма, а также его работа продемонстрирована на примере.

4.1 Построение алгоритма

В данном разделе изложен процесс построения алгоритма поиска путей в графе с заданными КС-ограничениями с использованием произведения Кронекера.

Пусть дан входной помеченный граф $\mathcal{G} = \langle V, E, L \rangle$ и входной КС-язык в качестве ограничений на пути в нём. Применимость матриц для описания графов уже была показана в предыдущей главе, однако для входных КС-ограничений в предложенных алгоритмах необходимо строить КС-грамматику и приводить её в ослабленную нормальную форму Хомского. Такое преобразование грамматики может привести как минимум к квадратичному увеличению её размера [36]. Особенностью алгоритма, предлагаемого в данной главе является использование рекурсивных автоматов для описания входных КС-ограничений без необходимости трансформации КС-грамматики.

Известно, что любая КС-грамматика может быть представлена с помощью рекурсивного автомата [39]. В свою очередь рекурсивный автомат, как и конечный автомат, может быть представлен в виде графа. Поэтому КС-ограничения также могут быть выражены с помощью таких объектов линейной алгебры, как матрицы. В таком случае, задачу поиска путей в графе с заданными КС-ограничениями можно решить с помощью нахождения пересечения рекурсивного автомата, соответствующего КС-ограничениям, и конечного автомата, соответ-

ствующего входному графу. Предлагаемый алгоритм основан на обобщении алгоритма пересечения конечных автоматов. Как было показано в разд. 2.2 для ограничений в виде регулярного языка, вычислять пересечение автоматов можно с помощью произведения Кронекера, применённого к матрицам из булевых декомпозициий матриц смежности для графовых представлений этих автоматов. Таким образом, используя операцию произведения Кронекера $\times$ и операцию $\bigvee$ поэлементного сложения булевых матриц, определённые над полукольцом $\langle\{0,1\},\vee,\wedge,0\rangle$, будет построена матрица, описывающая пересечение рекурсивного автомата для КС-ограничений и конечного автомата для входного графа. Однако такой рекурсивный автомат может иметь переходы с нетерминальными символами, которые имеют специальную логику рекурсивных вызовов, что требует добавления отдельного шага — вычисления транзитивного замыкания построенной матрицы для обработки этих символов. После вычисления транзитивного замыкания, в граф могут быть добавлены новые дуги (i, A, j) при обнаружении пути $i\pi j$, для которого $A \Rightarrow_G \lambda(\pi)$. После обновления графа процесс повторяется до сходимости.

Описанный алгоритм поиска путей в графе с заданными КС-ограничениями, использующий произведение Кронекера, представлен на листинге 7. Ключевая идея алгоритма заключается в итеративном вычислении произведения Кронекера для булевой декомпозиции матриц смежности $\mathcal{M}_1$ рекурсивного автомата и булевой декомпозиции матриц смежности $\mathcal{M}_2$ входного графа с последующим вычислением транзитивного замыкания результирующей матрицы. Наиболее трудоёмкими этапами в предложенном алгоритме являются вычисления произведения Кронекера и транзитивного замыкания. Кроме того, в предложенном алгоритме используется функция *getNonterminals*, которая по двум состояниям автомата R возвращает множество нетерминальных символов, которые соответствуют переходам между этими состояниями.

Результатом работы алгоритма, представленного на листинге 7, является булева декомпозиция матриц смежности $\mathcal{M}_2$ входного графа с добавленными дугами с нетерминальными символами в качестве меток, которые описывают достижимость вершин в соответствии с заданными КС-ограничениями, а также матрица M_3, которая содержит необходимую информацию для восстановления всех искомых путей. Далее покажем как предложенный алгоритм позволяет решать задачи поиска путей в графе с заданными КС-ограничениями.

Листинг 7 Алгоритм поиска путей в графе с заданными КС-ограничениями, использующий произведение Кронекера

```
1: function KroneckerBasedCFPQ(𝒢 = ⟨V, E, L⟩, G = ⟨N, Σ, P, S⟩)
2:     n ← |V|
3:     R ← рекурсивный автомат для грамматики G с m состояниями
4:     ℳ1 ← булева декомпозиция матрицы смежности для R с размерами матриц m × m
5:     ℳ2 ← булева декомпозиция матрицы смежности для 𝒢 с размерами матриц n × n
6:     C3, M3 ← пустые матрицы размера mn × mn
7:     for q ∈ {0, ..., m − 1} do
8:         for A ∈ getNonterminals(R, q, q) do
9:             for i ∈ {0, ..., n − 1} do
10:                ℳ2^A[i, i] ← 1
11:    while любая матрица из ℳ2 меняется do
12:        M3 ← ⋁_{M^A ∈ ℳ1 × ℳ2} M^A          ▷ Использование произведения Кронекера
13:        C3 ← transitiveClosure(M3)           ▷ Вычисление транзитивного замыкания
14:        for (i, j) | C3[i, j] = 1 do
15:            q1, q2 ← getStates(i, j, n)
16:            x, y ← getCoordinates(i, j, n)
17:            for A ∈ getNonterminals(R, q1, q2) do
18:                ℳ2^A[x,y] ← 1
19:    return ℳ2, M3
20: function getStates(i, j, n) ▷ Получение номеров состояний автомата R по индексам ячейки в матрице C3
21:    return ⌊i/n⌋ , ⌊j/n⌋
22: function getCoordinates(i, j, n)  ▷ Получение номеров вершин графа 𝒢 по индексам ячейки в матрице C3
23:    return i mod n, j mod n
```

Задача достижимости. Данная задача может быть решена с использованием множества матриц $\mathcal{M}_2$. А именно, в графе $\mathcal{G}$ существует путь из вершины i в вершину j, удовлетворяющий заданным КС-ограничениям, только если $\mathcal{M}_2^S[i, j] = 1$.

Задача поиска одного пути и задача поиска всех путей. Решить данные задачи можно с помощью матрицы M_3, которая для любого нетерминального символа $A \in N$ и для любой пары вершин (i, j) содержит достаточно информации для восстановления всех путей $i\pi j$ таких, что $A \Rightarrow_G \lambda(\pi)$. Поэтому на листинге алг. 8 представлен алгоритм восстановления всех путей в графе, соответствующих заданным КС-ограничениям. Стоит также отметить, что представленный алгоритм восстановления путей основан на методе поиска

в ширину, применённом на графе, соответствующем матрице M_3. А так как для данного метода обхода графа известен алгоритм [13], использующий операции линейной алгебры, то именно такие операции могут быть использованы для при реализации функции BFS обхода в ширину, которая возвращает пути из выделенной начальной вершины в конечную. Однако количество таких путей может быть бесконечным, поэтому при реализации функции BFS необходимо вычислять множество таких путей «лениво». Кроме того, для ограничения количества восстановленных путей в графе, возвращаемых алгоритмом, используется параметр k. При этом возвращаются k найденных путей с номерами вершин, образующих лексикографически наименьшие последовательности. Таким образом, предложенный алгоритм позволяет восстанавливать любое количество путей, удовлетворяющих заданным КС-ограничениям.

Листинг 8 Алгоритм восстановления всех путей в графе с заданными КС-ограничениями для алгоритма, основанного на произведении Кронекера

```
1: 𝓜₂ ← булева декомпозиция матрицы смежности результирующего графа алгоритма на листинге 7
2: M₃ ← результирующая матрица M₃ алгоритма на листинге 7
3: R ← рекурсивный автомат для входных КС-ограничений
4: 𝓜₁ ← булева декомпозиция матрицы смежности для R
5: function ExtractAllPaths(v_s, v_f, A, k)
6:     q_A^0 ← начальное состояние конечного автомата для нетерминала A
7:     F_A ← конечные состояния конечного автомата для нетерминала A
8:     paths ← ⋃_{q_A^f ∈ F_A} BFS(M₃, (q_A^0, v_s), (q_A^f, v_f))
9:     resultPaths ← ∅
10:    for path ∈ paths do
11:        currentPaths ← ∅
12:        for ((s_i, v_i), (s_j, v_j)) ∈ path do
13:            newEdges ← {(v_i, t, v_j) | M₂^t[v_i, v_j] ∧ M₁^t[s_i, s_j]}
14:            currentPaths ← currentPaths · newEdges                ▷ Конкатенация путей
15:            k' ← k − |resultPaths| − |currentPaths|
16:            newSubPaths ← ⋃_{{N | M₂^N[v_i,v_j] ∧ M₁^N[s_i,s_j]}} ExtractAllPaths(v_i, v_j, N, k')
17:            currentPaths ← currentPaths · newSubPaths             ▷ Конкатенация путей
18:        resultPaths ← resultPaths ∪ currentPaths
19:        if |resultPaths| ⩾ k then
20:            return TopK(resultPaths)
21:    return resultPaths
22: function TopK(paths)                                             ▷ Выбор k путей
23:    kPaths ← k путей из paths с наименьшими номерами вершин в лексикографическом порядке
24:    return kPaths
```

Таким образом, алгоритмы представленные на листингах 7 и 8 позволяют решить задачу поиска одного и задачу поиска всех путей в графе с заданными КС-ограничениями с использованием операции произведения Кронекера.

4.2 Корректность алгоритма

В данном разделе сформулированы и доказаны утверждения о корректности и завершаемости изложенного алгоритма.

Для доказательства завершаемости представленного алгоритма поиска путей в графе с заданными КС-ограничениями будет использовано определённое в разделе 3.2 отношение частичного порядка $\preceq_{rel}$ на элементах использованного булевого полукольца $\langle\{0,1\},\vee,\wedge,0\rangle$. Сперва докажем теорему о монотонности алгоритма, представленного на листинге 7. Далее для любых допустимых индексов (i,j) и для любого нетерминала $A \in N$, будем использовать следующие обозначения:

- $\mathcal{M}_2^{A,0}[i,j]$, $M_3^0[i,j]$ и $C_3^0[i,j]$ — для значений в ячейках $\mathcal{M}_2^A[i,j]$, $M_3[i,j]$ и $C_3[i,j]$ после инициализации матриц в строках 2–10 алгоритма, представленного на листинге 7;
- $\mathcal{M}_2^{A,k}[i,j]$, $M_3^k[i,j]$ и $C_3^k[i,j]$ — для значений в ячейках $\mathcal{M}_2^A[i,j]$, $M_3[i,j]$ и $C_3[i,j]$ после k исполнений цикла в строках 11–18, для $k \geqslant 1$.

Теорема 4.2.1 (Монотонность алгоритма)**.** Пусть $\mathcal{G} = \langle V, E, L\rangle$ — входной граф и $G = \langle N, \Sigma, P, S\rangle$ — входная КС-грамматика для алгоритма, представленного на листинге 7. Тогда для любых допустимых индексов (i,j) и для любого нетерминала $A \in N$, значения в ячейках $\mathcal{M}_2^A[i,j]$, $M_3[i,j]$ и $C_3[i,j]$ монотонно возрастают в соответствии с отношением частичного порядка $\preceq_{rel}$.

Доказательство. Необходимо доказать, что для любого $k \geqslant 1$, $\mathcal{M}_2^{A,k-1}[i,j] \preceq_{rel} \mathcal{M}_2^{A,k}[i,j]$, $M_3^{k-1}[i,j] \preceq_{rel} M_3^k[i,j]$ и $C_3^{k-1}[i,j] \preceq_{rel} C_3^k[i,j]$. Для $k = 1$ данное утверждение верно, так как все элементы матриц M_3 и C_3 равны $\perp = 0$, а значения элементов матриц $\mathcal{M}_2^A$ могут изменится лишь в строке 18 с 0 на 1. Для $k > 1$ данное утверждение верно, так как по свойствам определённых операций произведения Кронекера и транзитивного замыкания изменения некоторых значений элементов матриц $\mathcal{M}_2^{A,k-1}$ с 0 на 1 могут привести только к таким же

изменениям для некоторых элементов матриц M_3^{k-1} и C_3^{k-1}, что доказывает утверждение теоремы. □

Следствием доказанной теоремы является следующая теорема о конечности алгоритма, представленного на листинге 7.

Теорема 4.2.2 (Завершаемость алгоритма)**.** Алгоритм, представленный на листинге 7 завершается за конечное число шагов.

Доказательство. По теореме 4.2.1, значения в ячейках матриц из множества $\mathcal{M}_2^A$ монотонно возрастают. А так как матрицы и множества возможных значений элементов этих матриц конечны, то алгоритм, представленный на листинге 7 завершается за конечное число шагов. □

Кроме того, с помощью рассуждений, аналогичных рассуждениям в доказательствах леммы 3.2.2 и теоремы 3.2.3, может быть доказана следующая теорема.

Теорема 4.2.3 (Корректность алгоритма)**.** Пусть $\mathcal{G} = \langle V, E, L\rangle$ — входной граф и $G = \langle N, \Sigma, P, S\rangle$ — входная КС-грамматика для алгоритма, представленного на листинге 7. Тогда для любой пары вершин (i, j) графа $\mathcal{G}$, для любого нетерминала $A \in N$, значение в ячейке $\mathcal{M}_2^{A,k}[i, j] = 1$ только если существует путь $i\pi j$ такой, что $A \Rightarrow_G \lambda(\pi)$.

Доказательство. По индукции на минимальные высоты деревьев вывода, соответствующих рассматриваемым путям в графе. □

Следствием теоремы 4.2.3 является корректность матрицы M_3, возвращаемой алгоритмом из листинга 7 и описывающей пересечение конечного автомата для входного графа и рекурсивного автомата для входных КС-ограничений. Таким образом, корректен и предложенный алгоритм восстановления всех путей в графе, основанный на обходе в ширину матрицы M_3, что отображено в следующей теореме.

Теорема 4.2.4 (Корректность восстановления путей)**.** Пусть $\mathcal{G} = \langle V, E, L\rangle$ — входной граф, $G = \langle N, \Sigma, P, S\rangle$ — входная КС-грамматика и $(\mathcal{M}_2, M_3)$ — результат, возвращаемый алгоритмом, представленным на листинге 7. Тогда для любой пары вершин (i, j) и любого нетерминала $A \in N$ алгоритм, представленный на листинге 8 построит множество из заданного количества путей $i\pi j$

таких, что существует дерево вывода для строки $\lambda(\pi)$ из нетерминала A грамматики G, если такие пути существуют.

4.3 Временная сложность алгоритма

В данном разделе представлена оценка временной сложности предложенного алгоритма.

Предположим, что все операции над элементами использованных матриц вычисляются за $O(1)$ элементарных операций. Кроме того, оценим количество элементарных операций, необходимых в худшем случае для вычисления операций над булевыми матрицами, как:

- $O(n^2)$ — для операции $\bigvee$ поэлементного сложения двух булевых матриц размера $n \times n$,
- $O(m^2 n^2)$ — для операции $\times$ произведения Кронекера булевой декомпозиции матрицы смежности размера $m \times m$ и булевой декомпозиции матрицы смежности размера $n \times n$,
- $O(n^3 \log n)$ — для вычисления транзитивного замыкания булевой матрицы размера $n \times n$.

Стоит отметить, что вычисление транзитивного замыкания булевой матрицы за такое количество элементарных операций может быть выполнено с использованием техники быстрого возведения матрицы в степень [71].

Тогда справедлива следующая оценка временной сложности предложенного в этой главе алгоритма поиска путей в графе с заданными КС-ограничениями.

Теорема 4.3.1 (Оценка временной сложности алгоритма поиска путей, основанного на произведении Кронекера)**.** Пусть $\mathcal{G} = \langle V, E, L \rangle$ — входной граф, $G = \langle N, \Sigma, P, S \rangle$ — входная КС-грамматика и R — рекурсивный автомат для грамматики G с множеством состояний Q. Тогда для алгоритма, представленного на листинге 7, справедлива следующая оценка временной сложности: $O(|N||Q|^3|V|^5 \log(|Q||V|))$.

Доказательство. В строках 7–10 алгоритма для каждого состояния q производится чтение значений в ячейках $[q, q]$ в $|N|$ булевых матрицах размера

$|Q| \times |Q|$ с помощью функции *getNonterminals*, а затем для каждого полученного нетерминала A и каждой вершины графа i производится запись в ячейку $[i, i]$ соответствующей булевой матрицы. Поэтому в этих строках производится $O(|N||Q||V|)$ элементарных операций.

Далее рассмотрим цикл в строках 11–18. Алгоритм продолжает вычисления пока любая матрица из множества $\mathcal{M}_2$ меняется. Всего матриц в этом множестве, которые могут меняться — $|N|$ (они соответствуют нетерминальным символам грамматики G) и каждая из них имеет $|V|^2$ элементов. На каждой итерации могут измениться лишь некоторые нулевые значения ячеек этих матриц на значение 1. Поэтому максимальное количество итераций рассматриваемого цикла может быть достигнуто в случае, когда за каждую итерацию цикла изменяется значение лишь одной ячейки в одной матрице. Таким образом, максимальным количеством итераций рассматриваемого цикла является $|N||V|^2$. А на каждой итерации производится: вычисление произведения Кронекера за $O(|Q|^2|V|^2)$ операций; вычисление поэлементного сложения получившихся булевых матриц, соответствующих общим символам автомата и графа, а также нетерминальным символам, за $O(|Q|^2|V|^2)$ операций; вычисление транзитивного замыкания булевой матрицы размера $|Q||V| \times |Q||V|$ за $O(|Q|^3|V|^3 \log(|Q||V|))$ операций; и вычисления цикла в строках 14–18 за $O(|N||Q||V|)$ операций. Наиболее трудоёмким является вычисление транзитивного замыкания булевой матрицы размера $|Q||V| \times |Q||V|$, поэтому алгоритм завершит свою работу за $O(|N||V|^2(|Q|^3|V|^3 \log(|Q||V|)))$. Отсюда получаем заявленную оценку временной сложности алгоритма, представленного на листинге 7. □

Следует отметить что приведённая оценка временной сложности алгоритма, представленного на листинге 7, может быть существенно улучшена для разреженных графов. Также имеется возможность воспользоваться специальной техникой вычисления транзитивного замыкания [75], которая использует инкрементальность получающихся на итерациях алгоритма матриц и вычисляет информацию о достижимости только для необходимого множества вершин.

В данной работе временная сложность алгоритма, представленного на листинге 8, не приводится, так как она зависит от конкретной реализации функции *BFS*.

4.4 Пример

В данном разделе работа изложенного алгоритма продемонстрирована на примере, основанном на КС-языке $\mathcal{L} = \{a^n b^n \mid n \geqslant 1\}$. Рассмотрим граф $\mathcal{G}_1$, изображённый на рис. 1.1. Для алгоритма, изложенного в предыдущей главе было необходимо представить входные КС-ограничения в виде КС-грамматики в ослабленной нормальной форме Хомского. Пример такой грамматики для языка $\mathcal{L}$ был рассмотрен в разделе 3.4. Однако для алгоритма, представленного на листинге 7, приведение в нормальную форму необязательно. Поэтому рассмотрим грамматику с меньшим количеством правил и нетерминальных символов. Правила вывода такой КС-грамматики G имеют следующий вид.

$$\begin{aligned} 0: &\quad S \;\to\; a\ S\ b \\ 1: &\quad S \;\to\; a\ b \end{aligned}$$

Предложенный на листинге 7 алгоритм использует представление КС-грамматики в виде рекурсивного автомата. Такой автомат $R = \langle \{a, b\}, \{S\}, S, \{C_S\} \rangle$ для КС-грамматики G представлен на рис. 1.2.

Меткой стартового автомата C_S является стартовый нетерминал грамматики S, начальным состоянием этого автомата является состояние q_S^0, а множеством конечных состояний — $\{q_S^3\}$. Алгоритм будет использовать булевы декомпозиции $\mathcal{M}_1$ и $\mathcal{M}_2$ матриц смежности рекурсивного автомата и входного графа. Для более компактно представления в данном примере будем использовать сами матрицы смежности M_1 и M_2 для автомата R и графа $\mathcal{G}_1$, которые выглядят следующим образом (нулевые значения пропущены):

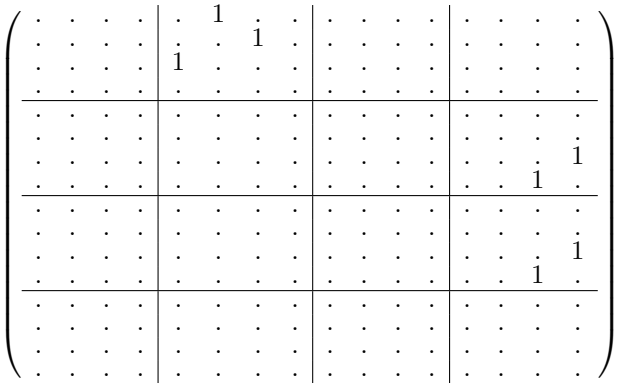
$$M_1 = \begin{pmatrix} . & \{a\} & . & . \\ . & . & \{S\} & \{b\} \\ . & . & . & \{b\} \\ . & . & . & . \end{pmatrix}, \qquad M_2^0 = \begin{pmatrix} . & \{a\} & . & . \\ . & . & \{a\} & . \\ \{a\} & . & . & \{b\} \\ . & . & \{b\} & . \end{pmatrix}.$$

В строках 7–10 алгоритм рассматривает все ε-переходы в автомате R и добавляет в матрицу M_2 информацию о пустых путях в графе. Однако в рассматриваемом примере рекурсивный автомат не содержит ε-переходов, поэтому в этих строках матрица M_2 не изменится.

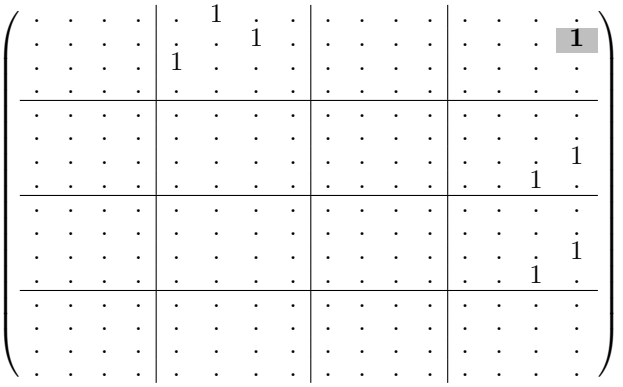
Далее алгоритм выполняет цикл в строках 11–18 пока матрица M_2 изменяется. Приведём значения матриц M_2, M_3 и C_3 на каждой итерации алгоритма.

На первой итерации цикла вычисляется произведение Кронекера $M_3^1 = M_1 \times M_2^0$ и транзитивное замыкание C_3^1, которые выглядят следующим образом (новые ненулевые значения выделены).

$M_3^1 =$



$C_3^1 =$



После вычисления транзитивного замыкания ячейка $C_3^1[1, 15]$ содержит ненулевое значение. Для того чтобы понять, что это означает для исходного графа $\mathcal{G}_1$ и рекурсивного автомата R, разберём используемые в алгоритме функции *getStates* и *getCoordinates*. С помощью функции *getStates* вычисляются значения состояний q_1 и q_2 рекурсивного автомата, которые соответствуют некоторому найденному пути в графе. Для вычисления x и y — начальной и конечной вершин этого пути, используется функция *getCoordinates*. Затем в матрицу M_2 добавляется информация о том, что существует путь в графе из вершины x в вершину y, метки на дугах которого образуют слово, выводимое из некоторого нетерминала на переходах из состояния q_1 в состояние q_2 рекурсивного автомата. Например, после первой итерации новое ненулевое значение в ячейке $C_3^1[1, 15]$ матрицы C_3^1 означает следующее.

- Состояния автомата $q_1 = 0$ и $q_2 = 3$, так как ячейка $C_3^1[1, 15]$ в верхнем правом блоке матрицы C_3 с координатами блока $(0, 3)$.
- Вершины $x = 1$ и $y = 3$, так как ячейка $C_3^1[1, 15]$ имеет именно такие координаты внутри своего блока.

- Функция *getNonterminals* возвращает множество $\{S\}$, так как это единственный нетерминал на переходах из состояния 0 в состояние 3 рекурсивного автомата R.
- Таким образом, существует путь в графе из вершины 1 в вершину 3, метки на дугах которого образуют слово, выводимое из нетерминала S.

В результате на первой итерации цикла в ячейку $M_2^1[1,3]$ добавляется нетерминал S. Обновлённая матрица M_2 и соответствующий обновлённый граф представлены на рис. 4.1.

$$M_2^1 = \begin{pmatrix} . & \{a\} & . & . \\ . & . & \{a\} & \{\mathbf{S}\} \\ \{a\} & . & . & \{b\} \\ . & . & \{b\} & . \end{pmatrix}$$

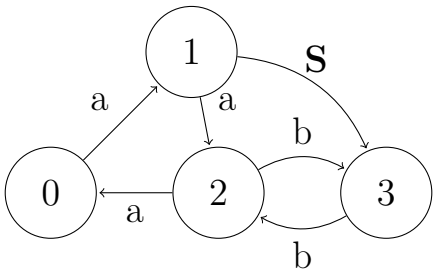


Рисунок 4.1 — Обновлённая матрица M_2 и соответствующий обновлённый граф после первой итерации алгоритма

На второй итерации цикла вычисленные матрицы M_3^2 и C_3^2 выглядят следующим образом:

$$M_3^2 = \left(\begin{array}{cccc|cccc|cccc|cccc} . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . \\ . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ \hline . & . & . & . & . & . & . & . & . & . & . & \mathbf{1} & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \end{array}\right)$$

$$C_3^2 = \left(\begin{array}{cccc|cccc|cccc|cccc} . & . & . & . & . & 1 & . & . & . & . & . & \mathbf{1} & . & . & \mathbf{1} & . \\ . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & 1 \\ . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ \hline . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & \mathbf{1} & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \end{array}\right).$$

Новые ненулевые значения матрицы C_3^2 появились в ячейках $[0,11]$, $[0,14]$ и $[5,14]$. Однако только ячейка с индексом $[0,14]$ соответствует состояниям q_1

$$M_2^2 = \begin{pmatrix} . & \{a\} & \{\mathbf{S}\} & . \\ . & . & \{a\} & \{S\} \\ \{a\} & . & . & \{b\} \\ . & . & \{b\} & . \end{pmatrix}$$

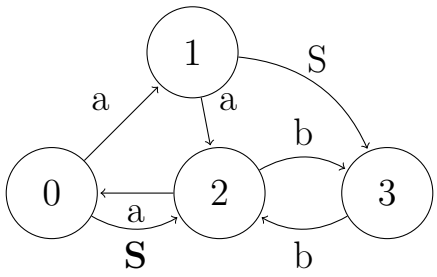


Рисунок 4.2 — Обновлённая матрица M_2 и соответствующий обновлённый граф после второй итерации

и q_2 автомата, между которыми есть хотя бы один переход по нетерминальному символу. Поэтому только значение в ячейке $[5, 14]$ повлияет на обновление матрицы M_2. Обновлённая матрица M_2 и соответствующий обновлённый граф после второй итерации представлены на рис. 4.2.

Матрицы M_2 и C_3, вычисленные на оставшихся итерациях цикла, представлены на рис. 4.3 и на рис. 4.5. В данном примере номер последней итерации цикла — 7, на которой в матрицу M_2 не добавляется новых ненулевых значений и алгоритм выходит из цикла.

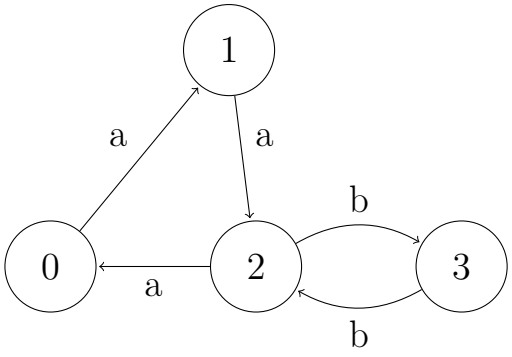
$$M_2^3 = \begin{pmatrix} . & \{a\} & \{S\} & . \\ . & . & \{a\} & \{S\} \\ \{a\} & . & . & \{b, \mathbf{S}\} \\ . & . & \{b\} & . \end{pmatrix} M_2^4 = \begin{pmatrix} . & \{a\} & \{S\} & . \\ . & . & \{a, \mathbf{S}\} & \{S\} \\ \{a\} & . & . & \{b, S\} \\ . & . & \{b\} & . \end{pmatrix}$$

$$M_2^5 = \begin{pmatrix} . & \{a\} & \{S\} & \{\mathbf{S}\} \\ . & . & \{a, S\} & \{S\} \\ \{a\} & . & . & \{b, S\} \\ . & . & \{b\} & . \end{pmatrix} M_2^6 = \begin{pmatrix} . & \{a\} & \{S\} & \{S\} \\ . & . & \{a, S\} & \{S\} \\ \{a\} & . & \{\mathbf{S}\} & \{b, S\} \\ . & . & \{b\} & . \end{pmatrix}$$

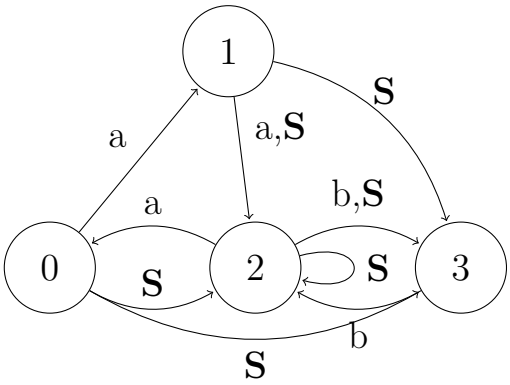
Рисунок 4.3 — Обновлённая матрица M_2 для итераций алгоритма с 3 по 6

Изначальный граф и граф со всеми добавленными дугами после работы алгоритма представлен на рис. 4.4.

Результатом работы алгоритма, представленного на листинге 7, являются матрицы M_2 и M_3. В матрице M_2 содержится вся информация о достижимости в графе в соответствии с заданными КС-ограничениями, которая отображена на рис. 4.4б. А последнее значение матрицы M_3 содержит информацию, достаточную для восстановления всех искомых путей.



а) Исходный граф $\mathcal{G}_1$



б) Результирующий граф

Рисунок 4.4 — Исходный граф $\mathcal{G}_1$ и граф, соответствующий результирующей булевой декомпозиции матрицы смежности $\mathcal{M}_2$

$$C_3^3 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & 1 & . \\
. & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & 1 & . & . & . & . & . & \mathbf{1} & . & . & . & . & \mathbf{1} \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
\hline
. & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & \mathbf{1} \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & 1 & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)$$

$$C_3^4 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & 1 & . \\
. & . & . & . & . & . & 1 & . & . & . & . & \mathbf{1} & . & . & \mathbf{1} & 1 \\
. & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
\hline
. & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & 1 & . \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & \mathbf{1} & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)$$

$$C_3^5 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & \mathbf{1} & 1 & . & . & 1 & \mathbf{1} \\
. & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & 1 & 1 \\
. & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
\hline
. & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & 1 & 1 & . & . & 1 & \mathbf{1} \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & 1 & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)$$

$$C_3^6 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & 1 & 1 & . & . & 1 & 1 \\
. & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & 1 & 1 \\
. & . & . & . & 1 & . & . & . & . & . & 1 & \mathbf{1} & . & . & \mathbf{1} & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
\hline
. & . & . & . & . & . & . & . & . & . & 1 & 1 & . & . & \mathbf{1} & 1 \\
. & . & . & . & . & . & . & . & . & . & 1 & 1 & . & . & 1 & 1 \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & 1 & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)$$

Рисунок 4.5 — Транзитивное замыкание C_3 на итерациях алгоритма с 3 по 6

4.5 Реализация

В данном разделе приведены детали реализации полученного алгоритма поиска путей в графе с заданными КС-ограничениями с использованием произведения Кронекера.

При реализации предложенного алгоритма также учитывались особенности реальных графов. В данном случае остаются применимы рассуждения из раздела разд. 3.5, в результате которых был сделан выбор использовать параллельные вычисления и разреженный формат для хранения матриц.

В предложенном алгоритме используется операция произведения Кронекера над булевыми матрицами, а также операция вычисления транзитивного замыкания матрицы. Алгоритм был реализован на CPU с использованием библиотеки SuiteSparse:GraphBLAS и на GPU с использованием библиотеки cuBool, в которых реализована операция произведения Кронекера и транзитивное замыкание может быть вычислено с использованием серии умножений булевых матриц (возведения матрицы M_3 в некоторую степень). В результате были получены следующие реализации:

- $KronAll_{CPU}$ — на языке `Python` с использованием пакета pygraphblas, являющегося обёрткой для библиотеки SuiteSparse:GraphBLAS;
- $KronAll_{GPU}$ — на языке `Python` с использованием пакета pycubool, являющегося обёрткой для библиотеки cuBool.

Стоит отметить, что предложенный алгоритм позволяет построить матрицы, которые содержат в себе информацию, достаточную для восстановления всех путей в графе, соответствующих заданным КС-ограничениям. Ответ на задачу достижимости может быть получен с использованием матриц $\mathcal{M}_2$, вычисленных с использованием разработанных реализаций. Для задач поиска одного и всех путей был реализован алгоритм восстановления искомых путей, представленный на листинге 8, по информации, содержащейся в матрицах $\mathcal{M}_2$ и M_3.

Реализации $KronAll_{CPU}$ и $KronAll_{GPU}$ также находятся в открытом доступе в рамках платформы *CFPQ_PyAlgo*.

Глава 5. Экспериментальное исследование

Завершаемость и корректность предложенных алгоритмов формально доказаны выше, однако их производительность требует экспериментальной оценки. При этом основной интерес представляет оценка и сравнение с существующими решениями на входных данных, близких к реальным.

5.1 Постановка экспериментов

В данной работе было показано, как предложенный подход может быть использован для получения алгоритмов поиска путей в графе с заданными КС-ограничениями и реализации этих алгоритмов. *Цель* этого экспериментального исследования — ответить на следующие *вопросы*.

[B1] Как показывают себя на практике полученные реализации для задачи достижимости в сравнении с существующими решениями на реальных данных?

[B2] Как показывают себя на практике полученные реализации для задач поиска одного и поиска всех путей в графе по сравнению с существующими решениями на реальных данных?

[B3] Как сказывается на производительности хранение в полученных реализациях информации о найденных путях в сравнении с полученными реализациями для задачи достижимости?

[B4] Как показывают себя на практике полученные реализации, не требующие преобразований входной КС-грамматики, по сравнению с другими предложенными реализациями?

Чтобы ответить на поставленные вопросы были выбраны следующие *метрики*.

[M1] Время работы реализаций.

[M2] Затрачиваемая реализациями память.

Для данного экспериментального исследования были выбраны графы, полученные из реальных RDF данных и программ, написанных на языках `C/C++`. Характеристики этих графов представлены в табл. 2 и табл. 3. В этих таблицах

приведено: номер графов для более компактного их упоминания; количество вершин и дуг графов; а также количество дуг с метками, которые в дальнейшем будут фигурировать в КС-ограничениях для этих графов.

Таблица 2 — Характеристики графов для анализа RDF данных [5]*

№	Граф	$\|V\|$	$\|E\|$	$\#subClassOf$	$\#type$
1	eclass	239,111	360,248	90,962	72,517
2	go	582,929	1,437,437	94,514	226,481
3	go_h	45,007	490,109	490,109	0
4	taxonomy	5,728,398	14,922,125	2,112,637	2,508,635
5	taxonomy_h	2,112,625	32,876,289	32,876,289	0

* $|V|$ — количество вершин графа, $|E|$ — количество дуг, а $\#subClassOf$ и $\#type$ — количества дуг с метками *subClassOf* и *type* соответственно.

Таблица 3 — Характеристики графов для статического анализа программ [76]*

№	Граф	$\|V\|$	$\|E\|$	$\#a$	$\#d$
6	apache_httpd_ptg	1,721,418	1,510,411	362,799	1,147,612
7	arch_after_inline	3,448,422	2,970,242	671,295	2,298,947
8	block_after_inline	3,423,234	2,951,393	669,238	2,282,155
9	crypto_after_inline	3,464,970	2,988,387	678,408	2,309,979
10	drivers_after_inline	4,273,803	3,707,769	858,568	2,849,201
11	fs_after_inline	4,177,416	3,609,373	824,430	2,784,943
12	init_after_inline	2,446,224	2,112,809	481,994	1,630,815
13	ipc_after_inline	3,401,022	2,931,498	664,151	2,267,347
14	kernel_after_inline	11,254,434	9,484,213	1,981,258	7,502,955
15	lib_after_inline	3,401,355	2,931,880	664,311	2,267,569
16	mm_after_inline	2,538,243	2,191,079	498,918	1,692,161
17	net_after_inline	4,039,470	3,500,141	807,162	2,692,979
18	postgre_sql_ptg	5,203,419	4,678,543	1,209,597	3,468,946
19	security_after_inline	3,479,982	3,003,326	683,339	2,319,987
20	sound_after_inline	3,528,861	3,049,732	697,159	2,352,573

* $|V|$ — количество вершин графа, $|E|$ — количество дуг, а $\#a$ и $\#d$ — количества дуг с метками a и d соответственно.

В данном экспериментальном исследовании будут выполнены два анализа: анализ RDF данных [5] и анализ указателей в программах на языках `C/C++` [76].

Оба этих анализа используют КС-ограничения в виде языков, описывающих различные правильные скобочные последовательности с двумя типами скобок. Правила вывода КС-грамматики G_{RDF} для анализа RDF данных имеют следующий вид.

$$\begin{array}{llll} 0: & S \to \overline{subClassOf}\ S\ subClassOf & 2: & S \to \overline{type}\ S\ type \\ 1: & S \to \overline{subClassOf}\ subClassOf & 3: & S \to \overline{type}\ type \end{array}$$

В то же время правила вывода КС-грамматики G_C для анализа указателей в программах на языках C/C++ выглядят следующим образом.

$$\begin{array}{llll} 0: & S \to \overline{d}\ V\ d & 4: & V_3 \to a\ V_2\ V_3 \\ 1: & V \to V_1\ V_2\ V_3 & 5: & V_1 \to \varepsilon \\ 2: & V_1 \to V_2\ \overline{a}\ V_1 & 6: & V_2 \to \varepsilon \\ 3: & V_2 \to S & 7: & V_3 \to \varepsilon \end{array}$$

КС-язык, порождаемый грамматикой G_{RDF}, описывает КС-ограничения, которые позволяют находить вершины в графе, находящиеся на одном уровне некоторой иерархии [5], а язык, порождаемый грамматикой G_C, позволяет находить в программах на языках C/C++ указатели, которые указывают на одну область памяти [2].

Стоит отметить, что в приведённых грамматиках используются не только метки $subClassOf$, $type$, a и d, но также метки $\overline{subClassOf}$, $\overline{type}$, $\overline{a}$ и $\overline{d}$. Дело в том, что для проведения указанных анализов необходимо рассматривать пути в графах, также содержащие обратные дуги. Поэтому для всех дуг (i, x, j) указанных графов, где $x \in \{subClassOf, type, a, d\}$ в явном виде добавлялись обратные дуги $(j, \overline{x}, i)$. Все графы и соответствующие грамматики доступны в наборе данных $CFPQ\_Data$[1].

Для проведения описанных анализов были использованы реализации, предложенные в данной диссертации и описанные в разд. 3.5 и в разд. 4.5, а также существующие решения, находящиеся в открытом доступе и которые автору удалось запустить. Полный список обозначений для используемых в сравнении реализаций представлен ниже.

- **MRC** — реализация $MtxReach_{CPU}$ предложенного алгоритма, использующего умножение матриц, для задачи достижимости.

[1] CFPQ_Data — набор данных для задач поиска путей в графе с заданными КС-ограничениями: https://jetbrains-research.github.io/CFPQ_Data/ (дата обращения: 14.01.2022).

- **MRG** — реализация $MtxReach_{GPU}$ на GPU предложенного алгоритма, использующего умножение матриц, для задачи достижимости.
- **MSC** — реализация $MtxSingle_{CPU}$ предложенного алгоритма, использующего умножение матриц, для задачи поиска одного пути.
- **MAC** — реализация $MtxAll_{CPU}$ предложенного алгоритма, использующего умножение матриц, для задачи поиска всех путей.
- **KAC** — реализация $KronAll_{CPU}$ предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.
- **KAG** — реализация $KronAll_{GPU}$ на GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.
- **Graspan**[2] — инструмент для высокопроизводительного параллельного статического анализа программ [76], написанный на языке `C++` и использующий CPU. Стоит отметить, что для инструмента *Graspan* также существует реализация на GPU, однако её запустить не удалось.
- **LL**[3] — реализация алгоритма достижимости в графе с заданными КС-ограничениями [24], основанного на алгоритме синтаксического анализа LL, написанная на языке `Go` и использующая CPU.
- $\mathbf{GLL_R}$ и $\mathbf{GLL_A}$[4] — реализации алгоритмов достижимости и поиска всех путей в графе с заданными КС-ограничениями [26], основанных на алгоритме синтаксического анализа GLL, написанные на языке `Java` и использующие CPU. В то время как реализация GLL_A строит SPPF для решения задачи поиска всех путей, реализация GLL_R предназначена для решения задачи достижимости и опускает это построение.

Кроме того, рассматриваемые реализации используют различные способы представления КС-ограничений, поэтому для некоторых реализаций КС-грамматики G_{RDF} и G_C приводились в ослабленную нормальную форму Хомского или для них строился соответствующий рекурсивный автомат. Стоит отметить, что приведение КС-грамматики G_{RDF} в ослабленную нормальную форму Хомского увеличило количество нетерминалов грамматики с 1 до 7 и количество

[2]Graspan — инструмент для статического анализа программ: https://github.com/Graspan/Graspan-C (дата обращения: 14.01.2022).

[3]LL — реализация алгоритма достижимости в графе с заданными КС-ограничениями, основанного на алгоритме LL: https://gitlab.com/ciromoraismedeiros/rdf-ccfpq (дата обращения: 14.01.2022).

[4]GLL — реализации алгоритмов достижимости и поиска всех путей в графе с заданными КС-ограничениями, основанные на алгоритме GLL: https://github.com/JetBrains-Research/GLL4Graph (дата обращения: 14.01.2022).

правил вывода с 4 до 10, а приведение в такую форму КС-грамматики G_C увеличило количество нетерминалов с 5 до 13 и количество правил вывода с 8 до 34.

Для сравнения производительности рассматриваемых реализаций вычислялось среднее время работы за 5 запусков, в соответствии с **[M1]**, а также потребляемая память, в соответствии с **[M2]**. Для задач поиска одного и всех путей в графе сравнивались только время работы и затраченная память на вычисление информации, достаточной для восстановления одного или всех путей в графе. Время восстановления найденных путей не измерялось, так как вид информации о восстановленных путях, а также выбор из многообразия способов её получения сильно зависят от области применения этих алгоритмов, что требует проведения отдельного исследования. Эксперименты проводились на компьютере под управлением ОС Ubuntu 18.04 с процессором Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz, DDR4 64GB оперативной памяти, файлом подкачки размера 64GB и видеокартой GeForce GTX 1070 с 8GB DDR5 памяти.

5.2 Результаты

[B1]. Чтобы ответить на первый вопрос было проведено сравнение всех реализаций, позволяющих получить информацию о достижимости в графе с заданными КС-ограничениями для анализа RDF данных и статического анализа программ. Так как все рассматриваемые реализации позволяют получить такую информацию, то все они участвовали в этом сравнении. Однако для алгоритма, основанного на произведении Кронекера, на текущий момент не существует специализированной для решения задачи достижимости реализации и для данного сравнения были выбраны реализации $KronAll_{CPU}$ и $KronAll_{GPU}$ для решения задачи поиска всех путей в графе. А из 2 реализаций, основанных на алгоритме GLL и из 4 предложенных реализаций, основанных на умножении матриц, были выбраны только реализации, специализированные для решения задачи достижимости. В соответствии с **[M1]** среднее время работы в секундах рассматриваемых реализаций для анализа RDF данных представлено в табл. 4, а для анализа указателей в программах на языках `C/C++` — в табл. 5. Также в таблицах указаны номера рассматриваемых графов и размер множества, являющегося ответом на поставленные задачи достижимости. То есть $\#result$ —

это количество пар вершин (i, j) таких, что существует хотя бы один путь из вершины i в вершину j, образующий слово из языка, порождаемого КС-грамматикой G_{RDF} или G_C. Кроме того, в таблицах выделено наименьшее время анализа для каждого графа.

Таблица 4 — Время работы в секундах алгоритмов достижимости в графах с заданными КС-ограничениями для анализа RDF данных [5]*

№	*#result*	*Graspan*	*LL*	GLL_R	*MRC*	*MRG*	*KAC*	*KAG*
1	90,994	2.5	9.3	1.5	**0.1**	**0.1**	0.3	0.2
2	640,316	12.8	46.0	5.6	1.2	**0.8**	3.2	3.1
3	588,976	0.9	51.4	3.7	**0.1**	0.2	0.2	0.2
4	151,706	3,938.0	253.1	45.5	**1.0**	**1.0**	6.0	3.9
5	5,351,657	3,817.1	3,023.8	OOT	**10.9**	OOM	11.7	OOM

* *#result* — размер результирующего множества; *Graspan* — инструмент [76] для статического анализа программ; *LL* — реализация алгоритма [24], основанного на алгоритме синтаксического анализа LL, GLL_R — реализация алгоритма [26] достижимости, основанного на алгоритме GLL; *MRC* и *MRG* — реализации на CPU и GPU предложенного алгоритма, использующего умножение матриц, для задачи достижимости; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

Результаты анализа RDF данных, представленные в табл. 4, показывают, что лучшими для этого анализа и выбранных графов являются реализации $MtxReach_{CPU}$ (*MRC*) и $MtxReach_{GPU}$ (*MRG*) для предложенного алгоритма, основанного на умножении матриц. Стоит отметить, что все четыре предложенные для данного анализа реализации $MtxReach_{CPU}$, $MtxReach_{GPU}$, $KronAll_{CPU}$ (*KAC*) и $KronAll_{GPU}$ (*KAG*) показывают лучшее время работы по сравнению с другими решениями. Реализации *LL* и GLL_R демонстрируют сравнимое с инструментом *Graspan* время работы, которому для проведения анализа на этих графах необходимо от примерно одной секунды до более чем часа. В то время как предложенные реализации позволяют проводить анализ на этих графах до 277 раз быстрее, чем лучшие из выбранных для сравнения существующих решений. Стоит отметить, что для данного анализа наиболее трудоёмким был анализ графа *taxonomy_h* с номером 5 и с более чем 5 миллионами пар вершин в результирующем множестве. Реализация GLL_R не завершила анализ за 5 часов, что отмечено в таблице с помощью сокращения OOT (Out Of Time), а реализации

Таблица 5 — Время работы в секундах алгоритмов достижимости в графах с заданными КС-ограничениями для статического анализа программ [76]*

№	*#result*	*Graspan*	*LL*	GLL_R	*MRC*	*MRG*	*KAC*	*KAG*
6	92,806,768	2,619.1	8,390.4	OOT	536.7	**135.0**	6,165.0	OOM
7	5,339,563	49.8	928.2	130.8	119.9	**34.5**	307.1	96.7
8	5,351,409	51.3	924.9	113.0	123.9	**34.4**	311.7	96.8
9	5,428,237	52.4	935.4	128.8	122.1	**34.7**	314.2	98.0
10	18,825,025	330.2	3,660.7	371.2	279.4	**69.8**	1,381.5	OOM
11	9,646,475	95.4	2,000.8	167.7	105.7	**49.6**	533.1	148.4
12	3,783,769	39.2	644.4	87.2	45.8	**24.6**	215.9	68.7
13	5,249,389	55.3	898.5	109.4	79.5	**34.0**	301.3	95.6
14	16,747,731	161.7	OOM	614.0	378.1	**104.8**	978.8	292.9
15	5,276,303	52.9	900.2	111.1	121.8	**34.1**	300.7	96.0
16	3,990,3	39.1	671.3	77.9	84.1	**25.5**	226.6	71.8
17	8,833,403	95.2	1,851.0	160.6	206.3	**55.2**	684.7	176.1
18	90,661,446	1,711.9	OOM	OOT	969.9	**170.4**	5,072.0	OOM
19	5,593,387	56.4	942.7	115.8	181.7	**35.1**	320.7	99.2
20	6,085,269	58.9	968.8	120.1	133.6	**36.1**	339.5	103.9

* *#result* — размер результирующего множества; *Graspan* — инструмент [76] для статического анализа программ; *LL* — реализация алгоритма [24], основанного на алгоритме синтаксического анализа LL, GLL_R — реализация алгоритма [26] достижимости, основанного на алгоритме GLL; *MRC* и *MRG* — реализации на CPU и GPU предложенного алгоритма, использующего умножение матриц, для задачи достижимости; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

$MtxReach_{GPU}$ и $KronAll_{GPU}$ завершили свою работу из-за нехватки памяти, которую необходимо выделить на GPU, что отмечено как OOM (Out Of Memory).

Для сравнения реализаций на более больших графах с миллионами вершин и дуг, а также на более трудоёмком анализе с десятками миллионов пар вершин в результирующих множествах, более показательны результаты, представленные в табл. 5. Для анализа указателей в программах на языках `C/C++` наилучший результат на всех данных показала GPU реализация $MtxReach_{GPU}$ предложенного алгоритма, основанного на умножении матриц. Такие результаты объясняются тем, что для трудоёмких вычислений на больших графах абсолютно оправданы затраты на обмен данными между CPU и

GPU для реализации высокопроизводительного анализа на GPU. Поэтому реализация $MtxReach_{GPU}$ позволяет производить данный анализ на выбранных графах до 19 раз быстрее, чем показывает лучшее из рассмотренных существующих решений — инструмент *Graspan*. Другая предложенная GPU реализация $KronAll_{GPU}$ демонстрирует большее время работы, так как эта реализация не специализирована для решения задачи достижимости и вычисляет избыточную информацию, которой достаточно для восстановления всех найденных путей. Если же рассматривать только реализации на CPU, то предложенная реализация $MtxReach_{CPU}$ показывает сравнимые с инструментом *Graspan* результаты, а на графах с самым трудоёмким анализом (с номерами 6, 10 и 18) позволяет добиться почти пятикратного увеличения скорости работы. Это является впечатляющим результатом, с учётом того, что инструмент *Graspan* специализируется на проведении статического анализа программ. Время работы реализации GLL_R сравнимо в данном анализе со временем работы реализации $MtxReach_{CPU}$, однако первой не хватило 5 часов для проведения двух самых трудоёмких анализов для графов с номерами 6 и 18. Наибольшее время работы показали реализации LL и $KronAll_{CPU}$. Однако стоит ещё раз отметить, что реализация $KronAll_{CPU}$ не специализирована на решении задачи достижимости. В данном анализе вычисление избыточной информации существенно сказалось на времени работы реализации $KronAll_{CPU}$, так как были найдены десятки миллионов пар достижимых вершин, а значит была вычислена информация для восстановления огромного количества путей, соответствующих заданным ограничениям.

В соответствии с **[M2]** для сравнения производительности рассмотренных реализаций были также измерены их затраты по памяти. Результаты этих измерений представлены в табл. 6 и в табл. 7, где выделены наименьшие затраты по памяти для каждого графа. Для решения задачи достижимости в процессе анализа RDF данных наименьшие затраты по памяти демонстрирует инструмент *Graspan*, что можно увидеть в табл. 6. Предложенные реализации потребляют до 6 раз больше памяти на данном анализе, чем инструмент *Graspan*, но существенно меньше, чем другие существующие решения LL и GLL_R. Для анализа указателей в программах на языках `C/C++` наименьшие затраты по памяти имеет реализация $MtxReach_{GPU}$ предложенного алгоритма, основанного на умножении матриц, что показано в табл. 7. А реализация $MtxReach_{CPU}$ потребляет сравнимый объём памяти с инструментом *Graspan*. Таким образом, для этого

анализа предложенные реализации позволяют снизить потребление памяти до 2 раз по сравнению с лучшим из существующих решений. Ожидаемо предложенные реализации $KronAll_{CPU}$ и $KronAll_{GPU}$ потребляют большее количество памяти, чем инструмент *Graspan*, из-за вычисления избыточной для решения задачи достижимости информации. Однако даже эти реализации потребляют существенно меньше памяти, чем другие существующие решения LL и GLL_R.

Таблица 6 — Затраты по памяти в мегабайтах алгоритмов достижимости в графах с заданными КС-ограничениями для анализа RDF данных [5]*

№	*#result*	*Graspan*	LL	GLL_R	*MRC*	*MRG*	*KAC*	*KAG*
1	90,994	**101**	470	1,263	240	307	279	357
2	640,316	**193**	1,406	1,192	468	727	468	829
3	588,976	**62**	431	3,177	263	387	266	573
4	151,706	**1,498**	15,200	9,877	3,229	1,651	3,229	2,463
5	5,351,657	**1,098**	20,395	OOT	6,804	OOM	6,804	OOM

* *#result* — размер результирующего множества; *Graspan* — инструмент [76] для статического анализа программ; LL — реализация алгоритма [24], основанного на алгоритме синтаксического анализа LL, GLL_R — реализация алгоритма [26] достижимости, основанного на алгоритме GLL; *MRC* и *MRG* — реализации на CPU и GPU предложенного алгоритма, использующего умножение матриц, для задачи достижимости; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

Таким образом, ответом на **[B1]** является следующее. Предложенные реализации для задачи достижимости по сравнению с лучшими из рассмотренных существующих решений позволяют:

- ускорить время анализа RDF данных до 277 раз, увеличив при этом до 6 раз объем потребляемой памяти;
- ускорить время анализа указателей в программах на языках `C/C++` до 19 раз, снизив при этом потребление памяти до 2 раз.

[B2]. Для ответа на второй вопрос аналогичные анализы RDF данных и программ на языках `C/C++` были проведены с помощью реализаций, вычисляющих информацию, достаточную для дальнейшего восстановления одного или всех найденных путей. Среди рассматриваемых существующих решений единственной реализацией, которая вычисляет такую информацию, является реализация

Таблица 7 — Затраты по памяти в мегабайтах алгоритмов достижимости в графах с заданными КС-ограничениями для статического анализа программ [76]*

№	*#result*	*Graspan*	*LL*	GLL_R	*MRC*	*MRG*	*KAC*	*KAG*
6	92,806,768	11,094	53,652	OOT	11,619	**5,585**	40,110	OOM
7	5,339,563	1,643	33,275	30,573	1,342	**863**	2,954	2,209
8	5,351,409	1,638	33,082	29,866	1,331	**849**	2,988	2,219
9	5,428,237	1,660	33,472	29,317	1,305	**849**	3,052	2,235
10	18,825,025	3,474	46,060	49,262	3,181	**2,861**	9,012	OOM
11	9,646,475	2,330	41,429	31,128	1,958	**1,099**	4,779	3,723
12	3,783,769	1,170	23,478	21,537	1,015	**687**	2,205	1,649
13	5,249,389	1,620	32,405	29,205	1,268	**845**	2,922	2,173
14	16,747,731	5,083	OOM	41,449	3,466	**1,959**	8,261	5,763
15	5,276,303	1,624	32,414	29,432	1,299	**845**	2,980	2,271
16	3,990,305	1,219	24,338	86,190	1,036	**691**	2,328	1,697
17	8,833,403	2,271	40,185	31,358	1,888	**1,111**	4,680	4,069
18	90,661,446	11,871	OOM	OOT	11,018	**5,297**	36,812	OOM
19	5,593,387	1,688	33,689	30,228	1,336	**857**	3,067	2,309
20	6,085,269	1,763	34,324	31,699	1,454	**887**	3,308	2,415

* *#result* — размер результирующего множества; *Graspan* — инструмент [76] для статического анализа программ; *LL* — реализация алгоритма [24], основанного на алгоритме синтаксического анализа LL, GLL_R — реализация алгоритма [26] достижимости, основанного на алгоритме GLL; *MRC* и *MRG* — реализации на CPU и GPU предложенного алгоритма, использующего умножение матриц, для задачи достижимости; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

GLL_A алгоритма поиска всех путей в графе с заданными КС-ограничениями, который основан на алгоритме GLL. В соответствии с **[M1]** среднее время работы в секундах рассматриваемых реализаций для анализа RDF данных представлено в табл. 8, а для анализа указателей в программах на языках `C/C++` — в табл. 9. Кроме того, в таблицах выделено наименьшее время анализа для каждого графа.

Таблица 8 — Время работы в секундах алгоритмов поиска одного и всех путей в графах с заданными КС-ограничениями для анализа RDF данных [5]*

№	*#result*	GLL_A	*MSC*	*MAC*	*KAC*	*KAG*
1	90,994	3.0	0.2	**0.1**	0.3	0.2
2	640,316	20.1	2.1	**0.5**	3.2	3.1
3	588,976	140.1	0.4	**0.2**	**0.2**	**0.2**
4	151,706	1,878.4	**3.0**	5.0	6.0	3.9
5	5,351,657	OOT	25.7	**11.0**	11.7	OOM

* *#result* — размер результирующего множества; GLL_A — реализация алгоритма [26] поиска всех путей, основанного на алгоритме GLL; *MSC* и *MAC* — реализации предложенного алгоритма, использующего умножение матриц, для задачи поиска одного пути и для задачи поиска всех путей соответственно; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

Результаты анализа RDF данных, представленные в табл. 8, показывают, что наименьшее время анализа выбранных графов имеют реализации $MtxSingle_{CPU}$ (*MSC*) и $MtxAll_{CPU}$ (*MAC*) для предложенного алгоритма, основанного на умножении матриц. Все рассматриваемые в данной таблице реализации позволяют решить задачу поиска одного пути в графе с заданными КС-ограничениями. А так как выбранной для сравнения существующей реализации GLL_A понадобилось более 5 часов для самого трудоёмкого анализа графа с номером 5, то предложенные реализации позволяют решить задачу поиска одного пути в рамках анализа RDF данных до 1636 раз быстрее в сравнении с этой реализацией. Аналогичный вывод верен и для задачи поиска всех путей в графе, так как наилучшее время анализа для графа с номером 5 показала реализация $MtxAll_{CPU}$, которая позволяет решить задачу поиска всех путей в графе.

Для задачи поиска одного пути в графе в рамках анализа указателей в программах на языках `C/C++` наилучшее время анализа показали предло-

Таблица 9 — Время работы в секундах алгоритмов поиска одного и всех путей в графах с заданными КС-ограничениями для статического анализа программ [76]*

№	*#result*	GLL_A	*MSC*	*MAC*	*KAC*	*KAG*
6	92,806,768	OOT	**1,611.5**	OOM	6,165.0	OOM
7	5,339,563	728.5	132.8	432.5	307.1	**96.7**
8	5,351,409	771.3	111.6	OOM	311.7	**96.8**
9	5,428,237	750.2	139.1	OOM	314.2	**98.0**
10	18,825,025	1,222.3	**699.1**	OOM	1,381.5	OOM
11	9,646,475	1,150.9	**135.6**	OOM	533.1	148.4
12	3,783,769	368.9	**53.4**	261.8	215.9	68.7
13	5,249,389	692.7	166.7	405.5	301.3	**95.6**
14	16,747,731	7,923.0	474.9	OOM	978.8	**292.9**
15	5,276,303	712.6	166.0	437.8	300.7	**96.0**
16	3,990,305	396.8	95.6	301.8	226.6	**71.8**
17	8,833,403	1,010.1	**145.8**	OOM	684.7	176.1
18	90,661,446	OOT	**2,024.0**	OOM	5,072.0	OOM
19	5,593,387	741.1	142.3	OOM	320.7	**99.2**
20	6,085,269	759.0	153.2	OOM	339.5	**103.9**

* *#result* — размер результирующего множества; GLL_A — реализация алгоритма [26] поиска всех путей, основанного на алгоритме GLL; *MSC* и *MAC* — реализации предложенного алгоритма, использующего умножение матриц, для задачи поиска одного пути и для задачи поиска всех путей соответственно; *KAC* и *KAC* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

женные реализации $MtxSingle_{CPU}$ и $KronAll_{GPU}$, что показано в табл. 9. Такие результаты объясняются тем, что реализация $MtxSingle_{CPU}$ является здесь единственной реализацией специализированной на задаче поиска одного пути в графе, а реализация $KronAll_{GPU}$ является единственной GPU реализацией в этом сравнении. Для задач поиска одного и поиска всех путей в графе в рамках анализа указателей в программах на языках `C/C++` предложенные реализации позволяют ускорить время анализа до 27 раз по сравнению с существующим решением GLL_A. Такой результат достигается с использованием GPU реализации $KronAll_{GPU}$ для анализа графа с номером 14.

В соответствии с **[M2]** для сравнения производительности рассмотренных реализаций для поиска одного и всех путей в графе были также измерены их затраты по памяти. Результаты этих измерений представлены в табл. 10 и в табл. 11, где выделены наименьшие затраты по памяти для каждого графа. Для решения задач поиска одного и поиска всех путей в графе в рамках анализа RDF данных наименьшие затраты по памяти демонстрирует предложенная реализация $MtxAll_{CPU}$, что можно увидеть в табл. 10. Исключением является граф с номером 1, для которого проводится наименее трудоёмкий анализ. Таким образом, для задач поиска одного и поиска всех путей в графе в рамках анализа RDF данных предложенные реализации потребляют до 152 раз меньше памяти, чем существующее решение. При этом стоит отметить, что предложенная реализация $MtxSingle_{CPU}$, специализированная на решении задач поиска одного пути в графе, потребляет большее количество памяти, чем реализация $MtxAll_{CPU}$ для решения задачи поиска всех путей в графе. Это можно объяснить особенностями реализации $MtxAll_{CPU}$, написанной на языке `C++` и использующей собственную обёртку на языке `Python`, а также малым количеством различных промежуточных вершин в элементах из множества *AllPathIndex* результирующих матриц, которые могут занимать меньше памяти, чем элементы из множества *PathIndex*. Также по результатам сравнения для анализа указателей в программах на языках `C/C++`, представленным в табл. 11, можно сделать вывод, что предложенные реализации потребляют сравнимое количество памяти с существующей реализацией GLL_A для задач поиска одного и поиска всех путей в графе. Однако стоит отметить, что для большинства рассматриваемых графов наименьшее потребление памяти продемонстрировала предложенная GPU реализация $KronAll_{GPU}$, а в реализации $MtxAll_{CPU}$ необходимая информация для восстановления всех найденных путей хранилась не столь компактно и анализ прервался из-за нехватки памяти.

По полученным результатам на **[B2]** можно ответить следующим образом. Предложенные реализации для задач поиска одного и поиска всех путей в графе по сравнению с рассмотренным существующим решением позволяют:

- ускорить время анализа RDF данных до 1636 раз, снизив при этом потребление памяти до 152 раз;
- ускорить время анализа указателей в программах на языках `C/C++` до 27 раз, потребляя при этом сравнимый объём памяти.

Таблица 10 — Затраты по памяти в мегабайтах алгоритмов поиска одного и всех путей в графах с заданными КС-ограничениями для анализа RDF данных [5]*

№	*#result*	GLL_A	*MSC*	*MAC*	*KAC*	*KAG*
1	90,994	**49**	257	200	279	357
2	640,316	649	545	**337**	468	829
3	588,976	30,444	290	**200**	266	573
4	151,706	9,108	3,805	**1,595**	3,229	2,463
5	5,351,657	OOT	8,058	**2,720**	6,804	OOM

* *#result* — размер результирующего множества; GLL_A — реализация алгоритма [26] поиска всех путей, основанного на алгоритме GLL; *MSC* и *MAC* — реализации предложенного алгоритма, использующего умножение матриц, для задачи поиска одного пути и для задачи поиска всех путей соответственно; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

[B3]. Чтобы ответить на третий вопрос необходимо сравнить полученные результаты предложенных реализаций для задачи достижимости, поиска одного и поиска всех путей в графе. Для наглядности результаты, полученные для предложенных реализаций, были вынесены в отдельные таблицы. В соответствии с **[M1]** среднее время работы в секундах предложенных реализаций для анализа RDF данных представлено в табл. 12, а для анализа указателей в программах на языках `C/C++` — в табл. 13.

Результаты анализа RDF данных, представленные в табл. 12, показывают, что предложенные реализации демонстрируют сравнимое время анализа выбранных графов. Можно отметить, что хранение в предложенных реализациях информации о найденных путях для решения задачи поиска одного пути замедляет время анализа до 3 раз, а для решения задачи поиска всех путей — до 4 раз. В свою очередь результаты анализа указателей в программах на языках `C/C++`, представленные в табл. 13, показывают, что наименьшее время работы среди предложенных реализаций имеет реализация $MtxReach_{GPU}$. Для решения задач поиска одного и поиска всех путей в графе в большинстве случаев наименьшее время работы демонстрирует реализация $KronAll_{GPU}$, которая замедляет анализ до 3 раз по сравнению с лучшей реализацией для задачи достижимости. Исключениями являются графы с номерами 6, 10 и 18, для которых прово-

Таблица 11 — Затраты по памяти в мегабайтах алгоритмов поиска одного и всех путей в графах с заданными КС-ограничениями для статического анализа программ [76]*

№	*#result*	GLL_A	*MSC*	*MAC*	*KAC*	*KAG*
6	92,806,768	OOT	**35,666**	OOM	40,110	OOM
7	5,339,563	6,043	2,651	62,423	2,954	**2,209**
8	5,351,409	5,874	2,651	OOM	2,988	**2,219**
9	5,428,237	6,250	2,676	OOM	3,052	**2,235**
10	18,825,025	**4,608**	8,332	OOM	9,012	OOM
11	9,646,475	8,629	4,214	OOM	4,779	**3,723**
12	3,783,769	4,242	1,964	62,404	2,205	**1,649**
13	5,249,389	5,992	2,589	62,384	2,922	**2,173**
14	16,747,731	**5,136**	7,156	OOM	8,261	5,763
15	5,276,303	6,094	2,632	62,421	2,980	**2,271**
16	3,990,305	4,545	2,073	62,453	2,328	**1,697**
17	8,833,403	9,346	4,201	OOM	4,680	**4,069**
18	90,661,446	OOT	**32,635**	OOM	36,812	OOM
19	5,593,387	6,157	2,759	OOM	3,067	**2,309**
20	6,085,269	6,842	2,991	OOM	3,308	**2,415**

* *#result* — размер результирующего множества; GLL_A — реализация алгоритма [26] поиска всех путей, основанного на алгоритме GLL; *MSC* и *MAC* — реализации предложенного алгоритма, использующего умножение матриц, для задачи поиска одного пути и для задачи поиска всех путей соответственно; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

дился наиболее трудоёмкий анализ и реализация $KronAll_{GPU}$ завершила свою работу из-за нехватки памяти. В процессе анализа этих графов предложенные реализации решали задачу поиска одного пути до 12 раз дольше, чем задачу достижимости, а задачу поиска всех путей — до 46 раз дольше.

В соответствии с **[M2]** предложенные реализации также требуют сравнения по количеству затраченной памяти. Результаты такого сравнения представлены в табл. 14 и в табл. 15. В процессе анализа RDF данных наименьшое потребление памяти продемонстрировала реализация $MtxAll_{CPU}$, которая позволяет решить задачу поиска всех путей в графе. Поэтому для данного анализа на выбранных графах с использованием предложенных реализаций не происхо-

Таблица 12 — Время работы в секундах предложенных алгоритмов поиска путей в графах с заданными КС-ограничениями для анализа RDF данных [5]*

№	*#result*	*MRC*	*MRG*	*MSC*	*MAC*	*KAC*	*KAG*
1	90,994	0.1	0.1	0.2	0.1	0.3	0.2
2	640,316	1.2	0.8	2.1	0.5	3.2	3.1
3	588,976	0.1	0.2	0.4	0.2	0.2	0.2
4	151,706	1.0	1.0	3.0	5.0	6.0	3.9
5	5,351,657	10.9	OOM	25.7	11.0	11.7	OOM

* *#result* — размер результирующего множества; *MRC* и *MRG* — реализации на CPU и GPU алгоритма, использующего умножение матриц, для задачи достижимости; *MSC* и *MAC* — реализации алгоритма, использующего умножение матриц, для задачи поиска одного пути и для задачи поиска всех путей; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

дит увеличение объёмов потребляемой памяти при переходе от решения задачи достижимости к решению задач поиска одного и поиска всех путей в графе. В то же время в процессе анализа указателей в программах на языках `C/C++` наименьшее потребление памяти продемонстрировала реализация $MtxReach_{GPU}$, решающая задачу достижимости. Для решения задач поиска одного и поиска всех путей в графе в большинстве случаев наименьшие затраты по памяти демонстрирует реализация $KronAll_{GPU}$, которая увеличивает эти затраты до 4 раз по сравнению с лучшей реализацией для задачи достижимости. Исключениями опять же являются три графа с наиболее трудоёмким анализом. В процессе анализа этих графов предложенные реализации использовали для решения задачи поиска одного пути до 6 раз больше памяти, чем для решения задачи достижимости, а для решения задачи поиска всех путей — до 7 раз больше памяти.

Таким образом, на **[В3]** можно ответить следующее. Хранение в полученных реализациях информации о найденных путях для решения задач поиска одного и поиска всех путей в графе по сравнению с предложенными реализациями для решения задачи достижимости влечёт за собой:

- замедление анализа RDF данных до 4 раз;
- замедление анализа указателей в программах на языках `C/C++` до 46 раз с потреблением до 7 раз большего объёма памяти.

Таблица 13 — Время работы в секундах предложенных алгоритмов поиска путей в графах с заданными КС-ограничениями для статического анализа программ [76]*

№	*#result*	*MRC*	*MRG*	*MSC*	*MAC*	*KAC*	*KAG*
6	92,806,768	536.7	135.0	1,611.5	OOM	6,165.0	OOM
7	5,339,563	119.9	34.5	132.8	432.5	307.1	96.7
8	5,351,409	123.9	34.4	111.6	OOM	311.7	96.8
9	5,428,237	122.1	34.7	139.1	OOM	314.2	98.0
10	18,825,025	279.4	69.8	699.1	OOM	1,381.5	OOM
11	9,646,475	105.7	49.6	135.6	OOM	533.1	148.4
12	3,783,769	45.8	24.6	53.4	261.8	215.9	68.7
13	5,249,389	79.5	34.0	166.7	405.5	301.3	95.6
14	16,747,731	378.1	104.8	474.9	OOM	978.8	292.9
15	5,276,303	121.8	34.1	166.0	437.8	300.7	96.0
16	3,990,305	84.1	25.5	95.6	301.8	226.6	71.8
17	8,833,403	206.3	55.1	145.8	OOM	684.7	176.1
18	90,661,446	969.9	170.4	2,024.0	OOM	5,072.0	OOM
19	5,593,387	181.7	35.1	142.3	OOM	320.7	99.2
20	6,085,269	133.6	36.1	153.2	OOM	339.5	103.9

* *#result* — размер результирующего множества; *MRC* и *MRG* — реализации на CPU и GPU алгоритма, использующего умножение матриц, для задачи достижимости; *MSC* и *MAC* — реализации алгоритма, использующего умножение матриц, для задачи поиска одного пути и для задачи поиска всех путей; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

Таблица 14 — Затраты по памяти в мегабайтах предложенных алгоритмов поиска путей в графах с заданными КС-ограничениями для анализа RDF данных [5]*

№	*#result*	*MRC*	*MRG*	*MSC*	*MAC*	*KAC*	*KAG*
1	90,994	240	307	257	200	279	357
2	640,316	468	727	545	337	468	829
3	588,976	263	387	290	200	266	573
4	151,706	3,229	1,651	3,805	1,595	3,229	2,463
5	5,351,657	6,804	OOM	8,058	2,720	6,804	OOM

* *#result* — размер результирующего множества; *MRC* и *MRG* — реализации на CPU и GPU алгоритма, использующего умножение матриц, для задачи достижимости; *MSC* и *MAC* — реализации алгоритма, использующего умножение матриц, для задачи поиска одного пути и для задачи поиска всех путей; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

[B4]. Чтобы ответить на последний вопрос необходимо сравнить полученные результаты предложенных реализаций для алгоритма, основанного на произведении Кронекера и не требующего преобразования входной КС-грамматики, с предложенными реализациями алгоритма, основанного на умножении матриц. Для данного сравнения также можно использовать результаты, представленные в табл. 12, табл. 13, табл. 14 и табл. 15. Так как для алгоритма, основанного на произведении Кронекера, не существует специализированных реализаций для задач достижимости и поиска одного пути в графе, то сравнивать будем реализации, решающие задачу поиска всех путей в графе. Кроме того, для задачи поиска всех путей в графе алгоритм, основанный на умножении матриц, реализован только на CPU, поэтому сравнивать будем только CPU реализации. Таким образом, для данного сравнения были выбраны реализации $MtxAll_{CPU}$ и $KronAll_{CPU}$.

В табл. 12 и табл. 14 можно увидеть, что несмотря на увеличение размеров грамматики после её преобразования в ослабленную нормальную форму Хомского, реализация $MtxAll_{CPU}$ не уступает в скорости анализа RDF данных и потребляет до 2 раз меньший объём памяти, чем реализация $KronAll_{CPU}$, не требующая преобразований входной КС-грамматики. Причиной этого является сравнительно небольшие размеры грамматики для данного анализа, а также

Таблица 15 — Затраты по памяти в мегабайтах предложенных алгоритмов поиска путей в графах с заданными КС-ограничениями для статического анализа программ [76]*

№	*#result*	*MRC*	*MRG*	*MSC*	*MAC*	*KAC*	*KAG*
6	92,806,768	11,619	5,585	35,666	OOM	40,110	OOM
7	5,339,563	1,342	863	2,651	62,423	2,954	2,209
8	5,351,409	1,331	849	2,651	OOM	2,988	2,219
9	5,428,237	1,305	849	2,676	OOM	3,052	2,235
10	18,825,025	3,181	2,861	8,332	OOM	9,012	OOM
11	9,646,475	1,958	1,099	4,214	OOM	4,779	3,723
12	3,783,769	1,015	687	1,964	62,404	2,205	1,649
13	5,249,389	1,268	845	2,589	62,384	2,922	2,173
14	16,747,731	3,466	1,959	7,156	OOM	8,261	5,763
15	5,276,303	1,299	845	2,632	62,421	2,980	2,271
16	3,990,305	1,036	691	2,073	62,453	2,328	1,697
17	8,833,403	1,888	1,111	4,201	OOM	4,680	4,069
18	90,661,446	11,018	5,297	32,635	OOM	36,812	OOM
19	5,593,387	1,336	857	2,759	OOM	3,067	2,309
20	6,085,269	1,454	887	2,991	OOM	3,308	2,415

* *#result* — размер результирующего множества; *MRC* и *MRG* — реализации на CPU и GPU алгоритма, использующего умножение матриц, для задачи достижимости; *MSC* и *MAC* — реализации алгоритма, использующего умножение матриц, для задачи поиска одного пути и для задачи поиска всех путей; *KAC* и *KAG* — реализации на CPU и GPU предложенного алгоритма, использующего произведение Кронекера, для задачи поиска всех путей.

сравнительно небольшая трудоёмкость самого анализа. Однако другие выводы можно сделать для более трудоёмкого анализа указателей в программах на языках C/C++. В табл. 13 и табл. 15 можно увидеть, что для большинства графов реализации $MtxAll_{CPU}$ не хватило памяти, а на остальных графах реализация $KronAll_{CPU}$ завершает анализ до полутора раз быстрее и потребляет до 28 раз меньше памяти. Существенное увеличение количества нетерминалов и правил вывода грамматики для такого трудоёмкого анализа с использованием алгоритма, основанного на умножении матриц, привело к необходимости хранить значительно большее количество булевых матриц с большим количеством ненулевых элементов и вычислять больше операций над этими матрицами.

Таким образом, на **[B4]** можно ответить следующее. Полученная CPU реализация, не требующая преобразований входной КС-грамматики, по сравнению с предложенной CPU реализацией алгоритма, основанного на умножении матриц, для задачи поиска всех путей в графе:

- не ускоряет анализ RDF данных и потребляет до 2 раз больше памяти из-за сравнительно небольших размеров грамматики и небольшой трудоёмкости самого анализа;
- ускоряет анализ указателей в программах на языках C/C++ до полутора раз и потребляет до 28 раз меньший объём памяти.

Выводы. Таким образом, полученные результаты экспериментального исследования позволяют сделать следующие выводы. Предложенный подход к задачам поиска путей в графе с заданными КС-ограничениями позволяет получать высокопроизводительные параллельные реализации, ускоряющие время анализа и потребляющие меньший объём памяти, в сравнении с существующими решениями на входных данных, близких к реальным.

5.3 Ограничения

Предлагаемые подход и алгоритмы поиска путей в графе с заданными КС-ограничениями накладывают ограничения на реализации этих алгоритмов. Данный раздел посвящён обсуждению этих ограничений.

Во-первых, для применения полученных реализаций требуется сформулировать необходимый анализ графов в виде задачи поиска путей с КС-ограничения. Для этого в классе КС-языков должен существовать язык, описывающий свойства путей необходимые для решения поставленной задачи анализа графов. Некоторые ограничения не могут быть описаны с использованием КС-языка. Для тех ограничений, которые удалось выразить в виде КС-языка, необходимо задать этот язык в виде, требуемом самим алгоритмом. Предложенные в данной работе алгоритмы используют КС-грамматики и рекурсивные автоматы в качестве представления КС-языков. А для алгоритмов, основанных на умножении матриц, такая грамматика должна быть преобразована в ослабленную нормальную форму Хомского. В процессе такого преобразования размеры КС-грамматики могут сильно увеличиться, что существенно скажется на производительности полученных реализаций.

Во-вторых, предложенные реализации сильно полагаются на разреженность входных графов. В случае анализа плотных графов больших размеров предложенные реализации либо будут показывать неудовлетворительное время работы, либо для этого анализа на используемой вычислительной системе может просто не хватить памяти. Также неэффективными себя показывают предложенные реализации на графах малого размера или для слишком простого анализа, когда время, затраченное на организацию параллельных вычислений, превышает или сравнимо со временем самого анализа.

Кроме того, после построения алгоритма поиска путей в графе с помощью предложенного подхода необходимо найти библиотеку линейной алгебры с реализованными необходимыми операциями или реализовать их самостоятельно. В случае использования в алгоритме стандартных операций (сложение, умножение, транспонирование и т.д.) над матрицами стандартных типов (булевого, целочисленного, с числами с плавающей точкой и т.д.) такую библиотеку найти не составляет труда. Однако в случае использования для более сложного анализа объектов линейной алгебры с некоторым пользовательским типом данных, с поиском подходящей библиотеки могут возникнуть проблемы. Если использовать одну из реализаций стандарта GraphBLAS, то необходимо убедиться, что использованный пользовательский тип данных в матрицах и векторах, а также операции над ними могут быть реализованы с использованием алгебраической структуры, схожей с полукольцом, которую позволяет построить стандарт GraphBLAS.

Глава 6. Сравнение и соотнесение

В данной главе представлено сравнение полученных результатов с основными существующими решениями задачи поиска путей в графе с заданными КС-ограничениями. Описание существующих решений представлено в разделе 1.5 данной работы.

В качестве инструментов, с которыми производилось сравнение, выбраны следующие инструменты, позволяющие анализировать графы с произвольными КС-грамматиками в качестве ограничений на пути: *LL* [24], основанный на алгоритме LL(1); GLL_R и GLL_A [26], основанные на алгоритме синтаксического анализа GLL; *Graspan* [76], который специализируется на статическом анализе программ. Сравнение производилось с предложенными реализациями $MtxReach_{CPU}$, $MtxReach_{GPU}$, $MtxSingle_{CPU}$ и $MtxAll_{CPU}$, основанными на умножении матриц, и реализациями $KronAll_{CPU}$ и $KronAll_{GPU}$, основанными на произведении Кронекера. Для сравнения были выбраны критерии, представленные в табл. 16.

В таблице 17 приведены основные результаты сравнения, которые позволяют сделать следующие выводы.

- На текущий момент не существует алгоритмов поиска путей в графе с заданными произвольными КС-ограничениями, использующих методы линейной алгебры, кроме предложенных в данной работе.
- Существующие инструменты в основном реализованы на CPU.
- В рамках данной работы была предложена первая реализация на GPU алгоритма, решающего задачи поиска одного и всех путей в графе с заданными КС-ограничениями.

Таблица 16 — Критерии сравнения инструментов для поиска путей в графе с заданными КС-ограничениями

Критерии	Название колонки в таблице с результатами сравнения 17	Описание
Отсутствие необходимости преобразования входной КС-грамматики	Без преобразований	Отсутствует ли необходимость преобразования входной КС-грамматики, например в нормальную форму, для применения алгоритма и соответствующего инструмента?
Восстановление путей	Восст. путей	Вычисляет ли алгоритм и соответствующий инструмент информацию, достаточную для восстановления одного или всех найденных путей в графе, соответствующих входным КС-ограничениям?
Использование GPU	GPU	Производит ли инструмент вычисления на GPU?
Использование методов линейной алгебры	Лин. алгебра	Используются ли в алгоритме и реализованы ли в соответствующем инструменте методы линейной алгебры?

Таблица 17 — Сравнение инструментов для поиска путей в графе с заданными КС-ограничениями

Инструмент	Без преобразований	Восст. путей	GPU	Лин. алгебра
LL	+	−	−	−
GLL_R	+	−	−	−
GLL_A	+	+*	−	−
Graspan	−**	−	+***	−
$MtxReach_{CPU}$	−	−	−	+
$MtxReach_{GPU}$	−	−	+	+
$MtxSingle_{CPU}$	−	+	−****	+
$MtxAll_{CPU}$	−	+	−****	+
$KronAll_{CPU}$	+	+	−	+
$KronAll_{GPU}$	+	+	+	+

* Рассматриваемая реализация алгоритма [26], основанного на алгоритме синтаксического анализа GLL, строит лес разбора SPPF, который хранит в себе информацию, достаточную для восстановления всех найденных путей в графе.

** Для использования инструмента *Graspan* требуется преобразовать входную КС-грамматику в ослабленную нормальную форму Хомского [76].

*** Для инструмента *Graspan* также существует реализация на GPU, которую автору не удалось запустить. В работе [76] утверждается, что такая реализация позволяет ускорить анализ указателей в некоторых программах на языках `C/C++` от 3.5 до 650 раз.

**** На данный момент алгоритм, основанный на умножении матриц, реализован на GPU только для задачи достижимости. Однако, например, для задач поиска одного или всех путей в графе такой алгоритм в будущем возможно удастся получить высокопроизводительную реализацию на GPU с использованием библиотеки CUSP или библиотеки GraphBLAST.

Заключение

Основные результаты работы заключаются в следующем.

1. Разработан подход к поиску путей в графе с заданными КС-ограничениями на основе методов линейной алгебры, который позволяет использовать теоретические и практические достижения линейной алгебры для решения данной задачи.
2. Разработан алгоритм, использующий предложенный подход и решающий задачи поиска путей в графе с заданными КС-ограничениями. Доказана завершаемость и корректность предложенного алгоритма. Получена теоретическая оценка сверху временной сложности алгоритма. Предложенный алгоритм использует операции над матрицами, которые позволяют применять широкий класс оптимизаций и дают возможность автоматически распараллеливать вычисления за счёт существующих библиотек линейной алгебры.
3. Разработан алгоритм поиска путей в графе с заданными КС-ограничениями, использующий предложенный подход и не требующий преобразования входной КС-грамматики. Доказана завершаемость и корректность предложенного алгоритма. Получена теоретическая оценка сверху временной сложности алгоритма. Предложенный алгоритм позволяет работать с произвольными входными КС-грамматиками без необходимости их преобразования, что позволяет избежать значительного увеличения размеров входной грамматики и увлечения времени работы алгоритма.
4. Предложенные алгоритмы реализованы с использованием параллельных вычислений. Проведено экспериментальное исследование разработанных алгоритмов на реальных RDF данных и графах, построенных для статического анализа программ. Было проведено сравнение полученных реализаций между собой, с существующими решениями из области статического анализа и с решениями, основанными на различных алгоритмах синтаксического анализа. Результаты сравнения показывают, что предложенные реализации для задачи достижимости позволяют ускорить время анализа до 2 порядков и потребляют до 2 раз меньше памяти по сравнению с существующими решениями, а для

задач поиска одного и поиска всех путей в графе позволяют ускорить время анализа до 3 порядков и до 2 порядков снизить потребление памяти.

На основе полученных реализаций была создана платформа *CFPQ_PyAlgo* для разработки и тестирования алгоритмов поиска путей в графе с заданными КС-ограничениями.

В рамках **рекомендации по применению результатов работы** в индустрии и научных исследованиях укажем, что разработанные подход и алгоритмы, полученные на его основе, применимы для поиска путей в графах с заданными КС-ограничениями с использованием методов линейной алгебры, а также что могут быть получены компактные, переносимые и высокопроизводительные реализации предложенных алгоритмов на различных параллельных вычислительных системах с использованием существующих библиотек линейной алгебры. Платформа полученная на основе предлагаемых реализаций, может быть использована для статического анализа программ [1; 2], сетевого анализа [5], в биоинформатике [3] и т.д. Кроме того, полученные реализации могут быть интегрированы с такими графовыми базами данных, как RedisGraph. В работе [28] были предложены прототипы таких реализаций. Однако для полноценной интеграции с графовой базой данных RedisGraph необходимо расширить язык запросов Cypher, используемый этой базой данных. И расширение синтаксиса этого языка запросов, позволяющее выразить КС-ограничения на пути в графе, было предложено[1].

Также определим **перспективы дальнейшей разработки тематики**, основной из которых является применение предложенного подхода и разработанных алгоритмов для создания специализированных инструментов для решения перечисленных прикладных задач. На основе предложенного подхода могут быть созданы алгоритмы, которые учитывают специфику графов и ограничений на пути в тех или иных прикладных областях. Например, для статического анализа программ может быть учтена структура графов, получаемых из программ на конкретном языке программирования, а также структура КС-грамматик, порождающих конкретный КС-язык для выбранного анализа (анализа указателей [2], поиска уязвимостей в программе [77] и т.д.).

[1]Предложение для расширения синтаксиса языка запросов openCypher: https://github.com/thobe/openCypher/blob/rpq/cip/1.accepted/CIP2017-02-06-Path-Patterns.adoc (date of access: 14.01.2022).

На практике задачи поиска путей в больших графах редко решаются без фиксации сравнительно небольшого множества возможных начальных и/или конечных вершин в искомых путях. Таким образом, часто информация о путях между любыми вершинами является избыточной. Поэтому ещё одним направлением, представляющим интерес, является модификация всех предложенных и создание новых алгоритмов для решения задач поиска путей в графе с дополнительными ограничениями на множества начальных и конечных вершин.

В данной работе предложенный алгоритм, основанный на умножении матриц, для задач поиска одного и всех путей в графе был реализован только на CPU. Для решения этих задач в алгоритме были использованы более сложные типы для элементов матриц, что приводит к существенному увеличению затрат по памяти и не позволяет получить высокопроизводительную реализацию на GPU. Поэтому необходимо проведение дальнейших исследований по оптимизации этого алгоритма и использованных в нём алгебраических структур. В дальнейшем, возможно, удастся получить высокопроизводительные реализации на GPU этого алгоритма с использованием библиотеки CUSP или библиотеки GraphBLAST.

Кроме того, известны некоторые задачи анализа графов, ограничения на пути в которых не могут быть выражены с помощью КС-языка. Например, такая задача возникает в области статического анализа программ, когда кроме анализа указателей производится анализ путей исполнений программы на корректную последовательность вызовов её подпрограмм и возвратов из них. Например, согласно работе [78] такая задача является неразрешимой, однако для аппроксимации результата такого анализа может быть использованы ограничения на пути в виде линейных конъюнктивных языков [79], которые принадлежат более широкому классу, чем контекстно-свободные. Таким образом, актуальным направлением является расширение предложенного подхода для решения задач поиска путей в графе с ограничениями, выраженными языками из более широких классов, чем класс КС-языков.

Список литературы

1. *Rehof, J.* Type-base flow analysis: from polymorphic subtyping to CFL-reachability [текст] / J. Rehof, M. Fähndrich // ACM SIGPLAN Notices. — 2001. — т. 36, № 3. — с. 54—66.
2. *Zheng, X.* Demand-driven alias analysis for C [текст] / X. Zheng, R. Rugina // Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. — 2008. — с. 197—208.
3. *Sevon, P.* Subgraph queries by context-free grammars [текст] / P. Sevon, L. Eronen // Journal of Integrative Bioinformatics. — 2008. — т. 5, № 2. — с. 157—172.
4. *Truong, Q.-D.* Graph Methods for Social Network Analysis [текст] / Q.-D. Truong, T. Dkaki, Q.-B. Truong //. т. 168. — 03.2016. — с. 276—286.
5. Context-free path queries on RDF graphs [текст] / X. Zhang [и др.] // International Semantic Web Conference. — Springer. 2016. — с. 632—648.
6. *Barrett, C.* Formal-language-constrained path problems [текст] / C. Barrett, R. Jacob, M. Marathe // SIAM Journal on Computing. — 2000. — т. 30, № 3. — с. 809—837.
7. *Reps, T.* Program analysis via graph reachability [текст] / T. Reps // Information and software technology. — 1998. — т. 40, № 11/12. — с. 701—726.
8. *Bradford, P. G.* Efficient exact paths for dyck and semi-dyck labeled path reachability [текст] / P. G. Bradford // 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON). — IEEE. 2017. — с. 247—253.
9. *Hellings, J.* Conjunctive Context-Free Path Queries. [текст] / J. Hellings // ICDT. — 2014. — с. 119—130.
10. *Hellings, J.* Explaining Results of Path Queries on Graphs [текст] / J. Hellings // Software Foundations for Data Interoperability and Large Scale Graph Data Analytics. — Springer, 2020. — с. 84—98.

11. *Koschmieder, A.* Regular path queries on large graphs [текст] / A. Koschmieder, U. Leser // International Conference on Scientific and Statistical Database Management. — Springer. 2012. — с. 177—194.

12. Answering regular path queries using views [текст] / D. Calvanese [и др.] // Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073). — IEEE. 2000. — с. 389—398.

13. *Kepner, J.* Graph algorithms in the language of linear algebra [текст] / J. Kepner, J. Gilbert. — SIAM, 2011.

14. An experimental study of context-free path query evaluation methods [текст] / J. Kuijpers [и др.] // Proceedings of the 31st International Conference on Scientific and Statistical Database Management. — 2019. — с. 121—132.

15. *Buluç, A.* Parallel breadth-first search on distributed memory systems [текст] / A. Buluç, K. Madduri // Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. — 2011. — с. 1—12.

16. Delta-stepping SSSP: From vertices and edges to GraphBLAS implementations [текст] / U. Sridhar [и др.] // 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). — IEEE. 2019. — с. 241—250.

17. GBTL-CUDA: Graph algorithms and primitives for GPUs [текст] / P. Zhang [и др.] // 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). — IEEE. 2016. — с. 912—920.

18. *Azad, A.* Parallel triangle counting and enumeration using matrix algebra [текст] / A. Azad, A. Buluç, J. Gilbert // 2015 IEEE International Parallel and Distributed Processing Symposium Workshop. — IEEE. 2015. — с. 804—811.

19. A comparative study on exact triangle counting algorithms on the gpu [текст] / L. Wang [и др.] // Proceedings of the ACM Workshop on High Performance Graph Processing. — 2016. — с. 1—8.

20. Mathematical foundations of the GraphBLAS [текст] / J. Kepner [и др.] // 2016 IEEE High Performance Extreme Computing Conference (HPEC). — IEEE. 2016. — с. 1—9.

21. *Spampinato, D. G.* Linear algebraic depth-first search [текст] / D. G. Spampinato, U. Sridhar, T. M. Low // Proceedings of the 6th ACM SIGPLAN International Workshop on Libraries, Languages and Compilers for Array Programming. — 2019. — с. 93—104.

22. *Valiant, L. G.* General context-free recognition in less than cubic time [текст] / L. G. Valiant // Journal of computer and system sciences. — 1975. — т. 10, № 2. — с. 308—315.

23. *Yannakakis, M.* Graph-theoretic methods in database theory [текст] / M. Yannakakis // Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. — 1990. — с. 230—242.

24. *Medeiros, C. M.* Efficient evaluation of context-free path queries for graph databases [текст] / C. M. Medeiros, M. A. Musicante, U. S. Costa // Proceedings of the 33rd Annual ACM Symposium on Applied Computing. — 2018. — с. 1230—1237.

25. *Santos, F. C.* A bottom-up algorithm for answering context-free path queries in graph databases [текст] / F. C. Santos, U. S. Costa, M. A. Musicante // International Conference on Web Engineering. — Springer. 2018. — с. 225—233.

26. *Grigorev, S.* Context-free path querying with structural representation of result [текст] / S. Grigorev, A. Ragozina // Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia. — 2017. — с. 1—7.

27. *Azimov, R.* Context-free path querying by matrix multiplication [текст] / R. Azimov, S. Grigorev // Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). — 2018. — с. 1—10.

28. Context-free path querying with single-path semantics by matrix multiplication [текст] / A. Terekhov [и др.] // Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). — 2020. — с. 1—12.

29. Context-free path querying by kronecker product [текст] / E. Orachev [и др.] // European Conference on Advances in Databases and Information Systems. — Springer. 2020. — с. 49—59.

30. *Azimov, R.* Context-free path querying with all-path semantics by matrix multiplication [текст] / R. Azimov, I. Epelbaum, S. Grigorev // Proceedings of the 4th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). — 2021. — с. 1—7.

31. *Azimov, R.* Context-Free Path Querying In Terms of Linear Algebra [текст] / R. Azimov. — 2021.

32. *Azimov, R.* Path Querying with Conjunctive Grammars by Matrix Multiplication [текст] / R. Azimov, S. Grigorev // Programming and Computer Software. — 2019. — т. 45, № 7. — с. 357—364.

33. *Азимов, Р. Ш.* Синтаксический анализ графов с использованием конъюнктивных грамматик [текст] / Р. Ш. Азимов, С. В. Григорьев // Труды Института системного программирования РАН. — 2018. — т. 30, № 2. — с. 149—166.

34. *Азимов, Р. Ш.* Алгоритм поиска всех путей в графе с заданными контекстно-свободными ограничениями с использованием матриц с множествами промежуточных вершин [текст] / Р. Ш. Азимов, С. В. Григорьев // Научно-технический вестник информационных технологий, механики и оптики. — 2021. — т. 21, № 4. — с. 499—505.

35. *Aho, A. V.* The theory of parsing, translation, and compiling [текст]. т. 1 / A. V. Aho, J. D. Ullman. — Prentice-Hall Englewood Cliffs, NJ, 1973.

36. *Hopcroft, J. E.* Introduction to automata theory, languages, and computation [текст] / J. E. Hopcroft, R. Motwani, J. D. Ullman // Acm Sigact News. — 2001. — т. 32, № 1. — с. 60—65.

37. *Rivera, F. F.* Euro-Par 2017: Parallel Processing: 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28–September 1, 2017, Proceedings [текст]. т. 10417 / F. F. Rivera, T. F. Pena, J. C. Cabaleiro. — Springer, 2017.

38. *Chomsky, N.* On certain formal properties of grammars [текст] / N. Chomsky // Information and control. — 1959. — т. 2, № 2. — с. 137—167.

39. Analysis of recursive state machines [текст] / R. Alur [и др.] // ACM Transactions on Programming Languages and Systems (TOPLAS). — 2005. — т. 27, № 4. — с. 786—818.

40. *Abiteboul, S.* Foundations of databases [текст]. т. 8 / S. Abiteboul, R. Hull, V. Vianu. — Addison-Wesley Reading, 1995.

41. *Verbitskaia, E.* Relaxed parsing of regular approximations of string-embedded languages [текст] / E. Verbitskaia, S. Grigorev, D. Avdyukhin // International Andrei Ershov Memorial Conference on Perspectives of System Informatics. — Springer. 2015. — с. 291—302.

42. Parser combinators for context-free path querying [текст] / E. Verbitskaia [и др.] // Proceedings of the 9th ACM SIGPLAN International Symposium on Scala. — 2018. — с. 13—23.

43. *Scott, E.* GLL parsing [текст] / E. Scott, A. Johnstone // Electronic Notes in Theoretical Computer Science. — 2010. — т. 253, № 7. — с. 177—189.

44. *Scott, E.* BRNGLR: a cubic Tomita-style GLR parsing algorithm [текст] / E. Scott, A. Johnstone, R. Economopoulos // Acta informatica. — 2007. — т. 44, № 6. — с. 427—461.

45. *Tomita, M.* LR parsers for natural languages [текст] / M. Tomita // 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics. — 1984. — с. 354—357.

46. *Hellings, J.* Querying for paths in graphs using context-free path queries [текст] / J. Hellings // arXiv preprint arXiv:1502.02242. — 2015.

47. *Horwitz, S.* Demand interprocedural dataflow analysis [текст] / S. Horwitz, T. Reps, M. Sagiv // ACM SIGSOFT Software Engineering Notes. — 1995. — т. 20, № 4. — с. 104—115.

48. *Reps, T.* Precise interprocedural dataflow analysis via graph reachability [текст] / T. Reps, S. Horwitz, M. Sagiv // Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. — 1995. — с. 49—61.

49. *Chatterjee, K.* Optimal Dyck reachability for data-dependence and alias analysis [текст] / K. Chatterjee, B. Choudhary, A. Pavlogiannis // Proceedings of the ACM on Programming Languages. — 2017. — т. 2, POPL. — с. 1—30.

50. *Dietrich, J.* Giga-scale exhaustive points-to analysis for java in under a minute [текст] / J. Dietrich, N. Hollingum, B. Scholz // Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications. — 2015. — с. 535—551.

51. An incremental points-to analysis with CFL-reachability [текст] / Y. Lu [и др.] // International Conference on Compiler Construction. — Springer. 2013. — с. 61—81.

52. Demand-driven points-to analysis for Java [текст] / M. Sridharan [и др.] // ACM SIGPLAN Notices. — 2005. — т. 40, № 10. — с. 59—76.

53. *Yan, D.* Demand-driven context-sensitive alias analysis for Java [текст] / D. Yan, G. Xu, A. Rountev // Proceedings of the 2011 International Symposium on Software Testing and Analysis. — 2011. — с. 155—165.

54. *Milanova, A.* CFL-reachability and context-sensitive integrity types [текст] / A. Milanova, W. Huang, Y. Dong // Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java platform: Virtual machines, Languages, and Tools. — 2014. — с. 99—109.

55. *Miao, H.* Understanding data science lifecycle provenance via graph segmentation and summarization [текст] / H. Miao, A. Deshpande // 2019 IEEE 35th International Conference on Data Engineering (ICDE). — IEEE. 2019. — с. 1710—1713.

56. *Bellman, R.* On a routing problem [текст] / R. Bellman // Quarterly of applied mathematics. — 1958. — т. 16, № 1. — с. 87—90.

57. *Ford, L. R.* Flows in networks [текст] / L. R. Ford, D. R. Fulkerson // Flows in Networks. — Princeton university press, 2015.

58. *Rote, G.* Path problems in graphs [текст] / G. Rote // Computational graph theory. — Springer, 1990. — с. 155—189.

59. *Rote, G.* A systolic array algorithm for the algebraic path problem (shortest paths; matrix inversion) [текст] / G. Rote // Computing. — 1985. — т. 34, № 3. — с. 191—219.

60. *Tarjan, R. E.* Fast algorithms for solving path problems [текст] / R. E. Tarjan // Journal of the ACM (JACM). — 1981. — т. 28, № 3. — с. 594—614.

61. *Tarjan, R. E.* A unified approach to path problems [текст] / R. E. Tarjan // Journal of the ACM (JACM). — 1981. — т. 28, № 3. — с. 577—593.

62. *Fletcher, J. G.* A more general algorithm for computing closed semiring costs between vertices of a directed graph [текст] / J. G. Fletcher // Communications of the ACM. — 1980. — т. 23, № 6. — с. 350—351.

63. *Floyd, R. W.* Algorithm 97: shortest path [текст] / R. W. Floyd // Communications of the ACM. — 1962. — т. 5, № 6. — с. 345.

64. *Warshall, S.* A theorem on boolean matrices [текст] / S. Warshall // Journal of the ACM (JACM). — 1962. — т. 9, № 1. — с. 11—12.

65. *Aho, A. V.* The design and analysis of computer algorithms [текст] / A. V. Aho, J. E. Hopcroft. — Pearson Education India, 1974.

66. Introduction to algorithms [текст] / T. H. Cormen [и др.]. — MIT press, 2009.

67. *Davis, T. A.* Algorithm 1000: SuiteSparse: GraphBLAS: Graph algorithms in the language of sparse linear algebra [текст] / T. A. Davis // ACM Transactions on Mathematical Software (TOMS). — 2019. — т. 45, № 4. — с. 1—25.

68. *Davis, T. A.* Graph algorithms via SuiteSparse: GraphBLAS: triangle counting and k-truss [текст] / T. A. Davis // 2018 IEEE High Performance extreme Computing Conference (HPEC). — IEEE. 2018. — с. 1—6.

69. *Davis, T.* Algorithm 9xx: SuiteSparse: GraphBLAS: graph algorithms in the language of sparse linear algebra [текст] / T. Davis // Submitted to ACM TOMS. — 2018.

70. *Yang, C.* GraphBLAST: A high-performance linear algebra-based graph framework on the GPU [текст] / C. Yang, A. Buluç, J. D. Owens // ACM Transactions on Mathematical Software (TOMS). — 2022. — т. 48, № 1. — с. 1—51.

71. *Baras, J. S.* Path problems in networks [текст] / J. S. Baras, G. Theodorakopoulos // Synthesis Lectures on Communication Networks. — 2010. — т. 3, № 1. — с. 1—77.

72. *Chen, G.-H.* On the parallel computation of the algebraic path problem [текст] / G.-H. Chen, B.-F. Wang, C.-J. Lu // IEEE Transactions on Parallel & Distributed Systems. — 1992. — т. 3, № 02. — с. 251—256.

73. *Lengauer, T.* Unstructured path problems and the making of semirings [текст] / T. Lengauer, D. Theune // Workshop on Algorithms and Data Structures. — Springer. 1991. — с. 189—200.

74. Gaussian elimination is not optimal [текст] / V. Strassen [и др.] // Numerische mathematik. — 1969. — т. 13, № 4. — с. 354—356.

75. *Ibaraki, T.* On-line computation of transitive closures of graphs [текст] / T. Ibaraki, N. Katoh // Information Processing Letters. — 1983. — т. 16, № 2. — с. 95—97.

76. Systemizing Interprocedural Static Analysis of Large-Scale Systems Code with Graspan [текст] / Z. Zuo [и др.] // ACM Trans. Comput. Syst. — New York, NY, USA, 2021. — июль. — т. 38, № 1/2. — URL: https://doi.org/10.1145/3466820.

77. Scalable and precise taint analysis for android [текст] / W. Huang [и др.] // Proceedings of the 2015 International Symposium on Software Testing and Analysis. — 2015. — с. 106—117.

78. *Zhang, Q.* Context-Sensitive Data-Dependence Analysis via Linear Conjunctive Language Reachability [текст] / Q. Zhang, Z. Su // Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. — Paris, France : Association for Computing Machinery, 2017. — с. 344—358. — (POPL 2017). — URL: https://doi.org/10.1145/3009837.3009848.

79. *Okhotin, A.* Conjunctive grammars [текст] / A. Okhotin // Journal of Automata, Languages and Combinatorics. — 2001. — т. 6, № 4. — с. 519—535.

Список рисунков

Список таблиц