

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Реализация TinkerPop инфраструктуры для WebGraph

Обучающийся / Student Стародубцев Андрей Игоревич

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования

Группа/Group M34391

Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика

Образовательная программа / Educational program Информатика и программирование 2018

Язык реализации ОП / Language of the educational program Русский

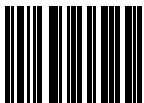
Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Руководитель ВКР/ Thesis supervisor Аксенов Виталий Евгеньевич, PhD, науки, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "ординарный доцент")

Консультант не из ИТМО / Third-party consultant Zacchiroli Stefano, Telecom Paris, Парижский политехнический институт, Франция / Telecom Paris, Polytechnic Institut of Paris, France, Профессор / Full Professor, Доктор наук / HDR, Professor

Обучающийся/Student


Документ подписан	
Стародубцев Андрей Игоревич	
20.05.2022	

(эл. подпись/ signature)

Стародубцев
Андрей
Игоревич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Аксенов Виталий Евгеньевич	
20.05.2022	

(эл. подпись/ signature)

Аксенов
Виталий
Евгеньевич

(Фамилия И.О./ name
and surname)

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS

Обучающийся / Student Стародубцев Андрей Игоревич

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования

Группа/Group М34391

Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика

Образовательная программа / Educational program Информатика и программирование 2018

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Тема ВКР/ Thesis topic Реализация TinkerPop инфраструктуры для WebGraph

Руководитель ВКР/ Thesis supervisor Аксенов Виталий Евгеньевич, PhD, науки, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "ординарный доцент")

Консультант не из ИТМО / Third-party consultant Zacchiroli Stefano, Telecom Paris, Парижский политехнический институт, Франция / Telecom Paris, Polytechnic Institut of Paris, France, Профессор / Full Professor, Доктор наук / HDR, Professor

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Требуется реализовать инфраструктуру фреймворка для обработки графов Apache TinkerPop, позволяющую обрабатывать графы, сжатые фреймворком WebGraph. Это позволит обрабатывать графы очень больших размеров (десятки миллиардов вершин) при помощи экспрессивного доменно-ориентированного языка для обхода графов Gremlin. Для реализации требуется разработать программный слой, связывающий модели данных TinkerPop и WebGraph, реализовать структурные интерфейсы TinkerPop для WebGraph, добавить поддержку различных подходов к хранению свойств вершин и ребер, оценить производительность реализации на представляющих интерес запросах в конкретном домене (архив репозитория Software Heritage).

TinkerPop — фреймворк изначально направленный на использование в графовых базах данных. При реализации для сжатых графов потребуется оптимизировать слой, связывающий два фреймворка, чтобы не препятствовать масштабируемости.

Поскольку WebGraph не предоставляет единого способа хранения свойств вершин/ребер, необходимо разработать абстракции, которые позволят TinkerPop получать доступ и вычислять значения этих свойств во время исполнения запросов.

Для анализа производительности потребуется реализовать на Gremlin ряд содержательных запросов в определенном домене и замерить время их выполнения. Сравнение с нативными реализациями запросов позволит оценить накладные расходы реализации.

Основными источниками являются документации TinkerPop (<https://tinkerpop.apache.org/docs/current/>), WebGraph (<https://webgraph.di.unimi.it/docs/>) и Software Heritage (<https://docs.softwareheritage.org/devel/swh-graph/>), а также наборы данных, на которых будут осуществляться проверки.

Дата выдачи задания / Assignment issued on: 31.01.2022

Срок представления готовой ВКР / Deadline for final edition of the thesis 15.05.2022

Характеристика темы ВКР / Description of thesis subject (topic)

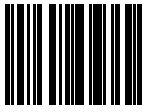
Название организации-партнера / Name of partner organization: Software Heritage

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: да / yes

СОГЛАСОВАНО / AGREED:

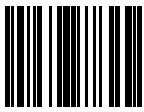
Руководитель ВКР/
Thesis supervisor

Документ подписан	
Аксенов Виталий Евгеньевич	
18.05.2022	

(эл. подпись)

Аксенов
Виталий
Евгеньевич

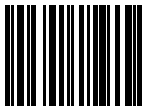
Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Стародубцев Андрей Игоревич	
19.05.2022	

(эл. подпись)

Стародубцев
Андрей
Игоревич

Руководитель ОП/ Head
of educational program

Документ подписан	
Станкевич Андрей Сергеевич	
01.06.2022	

(эл. подпись)

Станкевич
Андрей
Сергеевич

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Стародубцев Андрей Игоревич
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group M34391
Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика
Образовательная программа / Educational program Информатика и программирование 2018
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Реализация TinkerPop инфраструктуры для WebGraph
Руководитель ВКР/ Thesis supervisor Аксенов Виталий Евгеньевич, PhD, науки, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "ординарный доцент")
Консультант не из ИТМО / Third-party consultant Zacchiroli Stefano, Telecom Paris, Парижский политехнический институт, Франция / Telecom Paris, Polytechnic Institut of Paris, France, Профессор / Full Professor, Доктор наук / HDR, Professor

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Добавление возможности исполнения Gremlin запросов на графах, сжатых фреймворком WebGraph.

Задачи, решаемые в ВКР / Research tasks

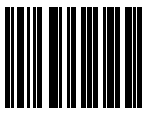
1) Реализация интерфейсов TinkerPop, позволяющих исполнять Gremlin запросы на графах, сжатых с использованием WebGraph. 2) Создание универсального механизма доступа к хранимым свойствам вершин и ребер, позволяющего использовать Gremlin запросы, опирающиеся на наличие и значения свойств. 3) Внедрение разработанной библиотеки в проект по архивации репозитория Software Heritage. 4) Проведение анализа производительности реализации и сравнение с нативными реализациями запросов.

Краткая характеристика полученных результатов / Short summary of results/findings

Разработана библиотека, позволяющая осуществлять немодифицирующие Gremlin запросы к любым графам, сжатым фреймворком WebGraph. Библиотека включает в себя поддержку различных источников свойств вершин и ребер, а также возможность определять собственные механизмы доступа к свойствам. Библиотека внедрена в проект по архивации данных систем контроля версий репозитория с открытым исходным кодом Software

Heritage. В этом домене сформулирован ряд запросов, и проведен анализ производительности библиотеки на этих запросах.

Обучающийся/Student

Документ подписан	
Стародубцев Андрей Игоревич	
20.05.2022	

(эл. подпись/ signature)

Стародубцев
Андрей
Игоревич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Аксенов Виталий Евгеньевич	
20.05.2022	

(эл. подпись/ signature)

Аксенов
Виталий
Евгеньевич

(Фамилия И.О./ name
and surname)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1. Обзор предметной области запросов к графовым структурам данных .	8
1.1. Язык для обхода графов Gremlin	8
1.2. Ограничения графовых баз данных.....	9
1.3. Сжатие графов	9
1.4. Ограничения WebGraph	9
1.5. Software Heritage	10
1.6. Проверка производительности	11
1.7. Постановка задачи	11
Выводы по главе 1	11
2. Предлагаемый подход к задаче добавления поддержки Gremlin для WebGraph.....	13
2.1. Разбиение на подзадачи	13
2.2. Реализация TinkerPop для WebGraph	13
2.2.1. Общая архитектура.....	13
2.2.2. Структурные интерфейсы и методы TinkerPop.....	13
2.2.3. Отличия графовых примитивов в WebGraph и TinkerPop ..	15
2.3. Работа со свойствами вершин и ребер графа.....	16
2.3.1. Свойства в TinkerPop	16
2.3.2. Свойства в WebGraph.....	16
2.4. Анализ производительности реализации	17
Выводы по главе 2	18
3. Реализация библиотеки, связывающей TinkerPop и WebGraph	19
3.1. Реализация TinkerPop	19
3.1.1. WebGraphGraph.....	19
3.1.2. WebGraphVertex	21
3.1.3. WebGraphEdge.....	21
3.1.4. WebGraphProperty	21
3.2. Реализация работы со свойствами	22
3.2.1. Провайдер свойств	22
3.2.2. Стандартный провайдер свойств	23
3.2.3. Стандартные свойства вершин	23
3.2.4. Стандартные свойства ребер.....	24

3.3.	Исполнитель запросов.....	26
3.4.	Использование разработанной библиотеки	27
3.5.	Внедрение результатов в проект Software Heritage.....	28
	Выводы по главе 3	29
4.	Сравнение разработанной библиотеки и нативного подхода	31
4.1.	Набор данных	31
4.2.	Формулировка запросов	31
4.2.1.	Самая ранняя ревизия для файла или директории.....	31
4.2.2.	Список файлов ревизии	32
4.2.3.	Дерево ревизий снимка	32
4.3.	Реализация запросов на Gremlin	33
4.3.1.	Самая ранняя ревизия для файла или директории.....	34
4.3.2.	Список файлов ревизии	34
4.3.3.	Дерево ревизий снимка	35
4.4.	Проведение тестирования	36
4.5.	Результаты тестирования	38
4.6.	Тестирование на других графах	41
4.7.	Преимущества библиотеки перед нативными запросами	43
	Выводы по главе 4	44
	ЗАКЛЮЧЕНИЕ	45
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	46
	ПРИЛОЖЕНИЕ А. Исполнитель Gremlin запросов.....	48
	ПРИЛОЖЕНИЕ Б. Пример использования библиотеки для добавления свойств	49
	ПРИЛОЖЕНИЕ В. Примеры нативных запросов.....	50

ВВЕДЕНИЕ

Графовое представление данных встречается все чаще [1], а увеличение размеров таких графов приводит к необходимости использовать системы, которые справляются с масштабируемостью. Для графов малых/средних размеров могут использоваться графовые базы данных, предоставляющие достаточную производительность, а также широкие возможности для анализа данных с помощью запросов, сформулированных на специальных доменно-ориентированных языках, таких как Gremlin [2]. В случае больших графов (десятки миллиардов вершин, сотни миллиардов ребер) решением является сжатие графов. Фреймворк WebGraph [3] является единственным подобным решением и позволяет добиться хорошей производительности, в свою очередь, требуя заметно меньше ресурсов по сравнению с графовыми базами данных [4–7]. Однако в данный момент такой подход предоставляет мало возможностей для анализа данных, поскольку способы доступа к вершинам и ребрам сжатого графа весьма лимитированы, и реализация запросов осуществляется вручную на каждый запрос, так как поддержки доменно-ориентированных языков для запросов WebGraph не предоставляет. Именно с такой проблемой столкнулась команда Software Heritage [8, 9], использующая фреймворк WebGraph для сжатия графа репозитория [7], и при консультации которой велась работа над данной задачей.

Иными словами, в данный момент пользователям, работающим с данными в графовом представлении, требуется выбрать: либо использовать графовую базу данных, чтобы получить возможность исполнять запросы на экспрессивном доменно-ориентированном языке, либо использовать сжатое представление графа, пожертвовав экспрессивностью языка запросов, взамен получив эффективную работу с памятью, и как следствие эффективное исполнение запросов. Цель данной работы — предоставить возможность пользоваться обоими преимуществами: формулировать запросы на экспрессивном доменно-ориентированном языке, сохранив превосходство сжатия графов в исполнении запросов. Для этого планируется добавить поддержку Gremlin запросов для графов, сжатых фреймворком WebGraph.

Первая глава содержит описание предметной области, а также постановку задачи и ее актуальность. Во второй главе рассматриваются шаги, необходимые для достижения поставленной цели, а также описан подход к верификации

решения. В третьей главе подробно описываются детали реализации основной части решения и показывается пример использования на практике. Четвертая глава показывает, каким образом был проведен анализ производительности решения, какие результаты были выявлены, а также сравнивает решение с текущим аналогом.

ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ ЗАПРОСОВ К ГРАФОВЫМ СТРУКТУРАМ ДАННЫХ

Многие задачи предполагают хранение данных в графовом представлении. Если пользователю необходимо извлечь некую информацию из этих данных, ему потребуется определить и выполнить некий запрос для этой структуры данных. Запрос может быть реализован нативно, то есть с использованием непосредственно методов лежащей в основе структуры. Такой подход не является гибким, так как требует ручной реализации на каждый отдельный случай. Альтернативой являются доменно-ориентированные языки для обхода графов, позволяющие формулировать гибкие запросы. В этой главе будет подробно описана данная предметная область, приведена решаемая проблема и аргументирована ее актуальность на конкретном примере.

1.1. Язык для обхода графов Gremlin

Gremlin — доменно-ориентированный язык для обхода графов, разработанный компанией Apache [2]. Является самым популярным языком подобного рода с открытым исходным кодом, о чем свидетельствует его широкая поддержка множеством графовых баз данных [10]. Действительно, основной областью использования Gremlin являются графовые базы данных, и это становится заметно при рассмотрении возможностей Gremlin. Несмотря на это, Gremlin позволяет гибко формулировать произвольные запросы для любых графов, вне зависимости от лежащей в основе системы.

В Gremlin запросах (с примером которого можно ознакомиться на листинге 1) пользователь может опираться как на структуру графа, то есть сами вершины и ребра, так и на свойства вершин и ребер. Свойства это дополнительная информация, хранящаяся в вершинах и ребрах. Примером такой информации может быть название аэропорта для графа рейсов, а примером свойства ребра может являться расстояние между аэропортами.

Листинг 1 – Пример запроса на Gremlin

```
g.V().has('code','LED')
    .repeat(out().simplePath())
    .until(has('code','SVO'))
    .path().by('code').limit(10)
```

Для исполнения запросов Gremlin использует фреймворк Apache TinkerPop [11]. Любая система имеет возможность реализовать инфраструктуру

ру Apache TinkerPop (англ. TinkerPop-enabled system), что позволит исполнять в ней Gremlin запросы. Примерами подобных систем являются упомянутые графовые базы данных.

1.2. Ограничения графовых баз данных

Предоставляя стандартную поддержку Gremlin, графовые базы данных дают возможность исполнять Gremlin запросы в своей системе — требуется адаптировать свою систему под одну из таких баз данных. Такое решение может подходить для графов малых/средних размеров. Однако, когда речь идет о больших графах (десятки миллиардов вершин, сотни миллиардов ребер), требуется масштабируемое решение. Графовые базы данных масштабируются за счет распределенных систем [4, 6], на создание и поддержание которых требуются большие ресурсы.

1.3. Сжатие графов

Другим подходом к обеспечению хорошей производительности при работе с большими графами является сжатие графов. Такой подход позволяет существенно сократить размер графа в памяти. Это, в свою очередь, дает возможность разместить весь граф в оперативной памяти одного относительно недорогого компьютера [7], что существенно повышает эффективность доступа к графу, и позволяет анализировать его без использования распределенной системы. Стоит отметить, что под сжатием графа понимается сжатие именно его структуры, а не свойств. Единственным решением подобного рода с открытым исходным кодом является фреймворк WebGraph [3].

1.4. Ограничения WebGraph

WebGraph позволяет эффективно хранить граф за счет сжатия его структуры, однако WebGraph не предоставляет поддержки Gremlin. Таким образом любой запрос к подобному графу требует реализации вручную, а WebGraph предоставляет весьма ограниченные способы доступа к структуре графа. Помимо этого, WebGraph не предоставляет универсального способа для работы со свойствами вершин и ребер, работа с ними во многом становится обязанностью пользователя. При этом многие представляющие интерес запросы зависят от значений свойств вершин и ребер.

1.5. Software Heritage

Software Heritage — проект, занимающийся архивированием данных систем контроля версий репозиториях с открытым исходным кодом. Архив представляет собой единый граф. Графовое представление обусловлено устройством распределенных систем контроля версий, таких как git [12], базирующихся на дереве хешей (точнее на ациклическом ориентированном графе). Структура графа приведена на рисунке 1.

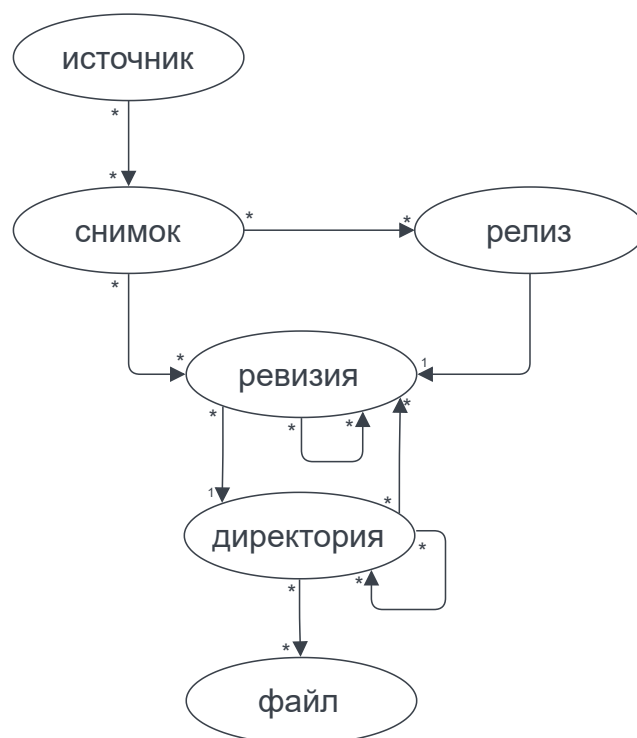


Рисунок 1 – Структура архива Software Heritage

Единый граф архива содержит более 20 миллиардов вершин и более 230 миллиардов ребер [13]. При таком крупном размере графа эффективное исполнение запросов на нем возможно при сжатии структуры графа, что позволяет разместить весь граф в менее чем ста гигабайтах оперативной памяти [7]. При этом для сжатия структуры используется упомянутый ранее фреймворк WebGraph.

Архив Software Heritage содержит большое количество информации, анализ которой может представлять интерес. Веб-сервер Software Heritage предоставляет доступ к фиксированному набору запросов на графе, позволяющих анализировать информацию, хранящуюся в графе [14]. Как было описано в разделе 1.4, WebGraph предоставляет возможность осуществлять запросы к сжатому графу только в нативном виде — непосредственно используя методы,

предоставляющие доступ к структуре графа. Это означает, что каждый предоставленный веб-сервером запрос потребовал непосредственной реализации на стороне сервера, а реализация нового запроса потребует вновь использовать лимитированные методы WebGraph. Для того чтобы упростить реализацию новых запросов к графу Software Heritage, а также стандартизировать подход к их реализации (вместо изучения методов WebGraph и реализации поверх них, может быть использован стандартный доменно-ориентированный язык), команда хочет изучить возможность использования языка Gremlin для этих нужд [9]. В случае успеха это позволит обрабатывать большой граф Software Heritage с удобством экспрессивного высокоуровневого языка, при этом сохранив эффективность исполнения запросов, достигаемую сжатием графа.

1.6. Проверка производительности

Поскольку TinkerPop предоставляет широкие возможности для обхода графов и в основном направлен на работу с графовыми базами данных, а WebGraph, в свою очередь предоставляет весьма лимитированный набор методов, предоставляющих доступ к структуре графа, фокусируясь на оптимизации памяти, следует проверить, не внесет ли их связка проблем с производительностью, и убедиться, что решение имеет практическую пользу. Для этого потребуется провести анализ производительности, собрав данные об эффективности исполнения запросов, и сравнив их с нативными реализациями.

1.7. Постановка задачи

Целью работы является реализация TinkerPop инфраструктуры для фреймворка для сжатия графов WebGraph. Это позволит выполнять сложные запросы на больших графах, используя язык Gremlin. Кроме того, необходимо добавить возможность указывать сторонние источники свойств вершин/ребер. После этого требуется сформулировать ряд запросов, анализ производительности исполнения которых покажет, насколько данный подход применим на практике.

Выводы по главе 1

В первой главе была описана предметная область работы. Было показано, какая проблема существует, приведен пример случая, в котором решение

этой проблемы будет актуально, а также разобраны альтернативные способы решения, и почему в этом случае они не подходят.

ГЛАВА 2. ПРЕДЛАГАЕМЫЙ ПОДХОД К ЗАДАЧЕ ДОБАВЛЕНИЯ ПОДДЕРЖКИ GREMLIN ДЛЯ WEBGRAPH

В данной главе будет описана общая структура решения, проведено разбиение на подзадачи, подробнее описаны детали каждой подзадачи.

2.1. Разбиение на подзадачи

Исходя из постановки задачи можно выделить три отдельные подзадачи:

- а) реализовать интерфейсы Apache TinkerPop, превращающие WebGraph в TinkerPop-enabled систему;
- б) реализовать возможность указывать источники и способы доступа к свойствам вершин и ребер для использования этих свойств в запросах;
- в) провести анализ производительности полученной реализации;

Далее каждая из подзадач по отдельности будет рассмотрена подробнее.

2.2. Реализация TinkerPop для WebGraph

В данном разделе описывается общая архитектура решения, а также предложен метод взаимодействия между представлениями графа в TinkerPop и WebGraph.

2.2.1. Общая архитектура

Поскольку данная реализация нацелена на случай, в котором пользователь уже использует фреймворк WebGraph, а значит располагает представлением сжатого графа, и имеет возможность исполнять на нем нативные запросы, однако хочет получить возможность исполнять запросы и на Gremlin, следует рассматривать реализацию как библиотеку, получающую в том или ином виде сжатый WebGraph граф, и предоставляющую таким образом возможность исполнять на нем запросы на Gremlin, делегируя описанные выше методы лежащему в основе сжатому графу с добавлением требуемой пред и постобработки аргументов и результатов.

2.2.2. Структурные интерфейсы и методы TinkerPop

Как было описано в разделе 1.1, для получения возможности исполнять Gremlin запросы в собственной системе, требуется реализовать интерфейсы фреймворка Apache TinkerPop. Конкретнее, нужно реализовать интерфейсы

пакета `/structure`, которые предоставляют абстракции для основных примитивов графовой модели данных — граф, вершина, ребро. Также, в этом пакете содержатся интерфейсы и методы для работы со свойствами, которые будут подробнее разобраны в разделе 2.3.

Рассмотрим основные интерфейсы пакета `/structure`:

- `Graph` — основной интерфейс графа. Экземпляр позволяет получать доступ к вершинам и ребрам по их идентификатору, либо ко всем вершинам и ребрам графа;
- `Vertex` — интерфейс вершины графа. Позволяет получить входящие и исходящие вершины и ребра, а также доступные свойства по их ключу;
- `Edge` — интерфейс ребра графа. Позволяет получить входящие и исходящие вершины, а также доступные свойства по их ключу;
- `Property` — интерфейс свойства вершины и ребра. Предоставляет доступ к значениям свойств;

Помимо описанных методов, `TinkerPop` предоставляет группы методов, поддержка которых реализовываться не будет:

- модифицирующие запросы — `TinkerPop` поддерживает запросы, модифицирующие граф. В данной реализации такие методы поддерживаться не будут, так как `WebGraph` не предоставляет возможности модифицировать граф во время работы с ним;
- метасвойства вершин — `TinkerPop` поддерживает свойства вершин, являющиеся полноценными элементами графа, то есть имеющих свой уникальный идентификатор и свойства. В данной реализации такая возможность поддерживаться не будет, так как стандартной поддержки как мета, так и обычных свойств вершин `WebGraph` не предоставляет;
- методы, специфичные для графовых баз данных — `TinkerPop` поддерживает методы, связаны со случаем использования в графовой базе данных, например `transaction()`. В данной реализации такая возможность поддерживаться не будет, так как в описываемом случае реализация основывается не на графовой базе данных;

Не стоит считать, что наличие подобных ограничений описываемого случая и, как следствие, отсутствие реализации указанных групп методов означает недостаточную совместимость `TinkerPop` и `WebGraph`. Как упоминалось в разделе 1.1, несмотря на то, что `TinkerPop` действительно в большей мере на-

правлен на работу с графовыми базами данных, где указанные группы методов оправданы, TinkerPop является фреймворком общего назначения для любых графовых систем, и отсутствие реализации указанных групп методов является ожидаемым, что подтверждается наличием стандартных исключений для каждой описанной ранее группы методов, информирующих об отсутствии реализации этой группы в данной системе.

2.2.3. Отличия графовых примитивов в WebGraph и TinkerPop

Как было описано в разделе 2.2.2, TinkerPop предоставляет абстракции для вершин и ребер, и работает с их экземплярами. WebGraph, в свою очередь, предоставляет доступ только к вершинам, ссылаясь на их уникальный числовой идентификатор — индекс. Располагая индексом вершины можно получить итератор по индексам вершин-соседей. При этом отдельной абстракции для ребер в WebGraph нет. Это позволяет эффективно работать со структурой графа в памяти, однако усложняет работу с ней пользователю. Отличие в представлениях этих примитивов означает, что для реализации взаимодействия потребуется создавать объекты-обертки. Здесь можно рассмотреть два подхода:

- а) сопоставить каждой вершине или ребру в WebGraph единственный объект, который будет возвращаться при любом доступе к данной вершине или ребру;
- б) возвращать временные объекты при доступе к данной вершине или ребру — нет гарантии, что два различных запроса к одной вершине или ребру вернут один и тот же объект, однако гарантируется, что объекты будут равны и будут обладать одинаковыми свойствами;

Первый вариант подразумевает создание единого реестра объектов для получения гарантии идентичности возвращаемых объектов при различных запросах. Учитывая цель сохранить преимущество в использовании памяти, получаемое за счет сжатия графа, этот подход не является лучшим выбором, так как создает дополнительную нагрузку на память, которая при участии в обходе достаточного числа вершин, станет существенной проблемой. Второй подход, в свою очередь, удобен своей гибкостью. Оставляя возможность создавать новые объекты при каждом новом обращении к вершине, он не исключает возможности внесения оптимизаций для уменьшения общего числа аллокаций новых объектов.

2.3. Работа со свойствами вершин и ребер графа

Вершины и ребра в графе могут обладать свойствами. Примером свойства вершины может быть название аэропорта для графа рейсов, а свойством ребра может являться расстояние между аэропортами. Многие запросы к графовой структуре данных представляют интерес именно в привязке к свойствам и их значениям, так как без их учета запрос может опираться только на структуру графа. Примером запроса на графе рейсов, зависящего от свойств вершин, может являться самый ранний рейс по выбранному направлению. В данном случае время вылета будет свойством вершины, обозначающей рейс.

2.3.1. Свойства в TinkerPop

Gremlin предоставляет возможность формулировать запросы, зависящие от свойств. Свойства устроены как пара ключ-значение, где ключ является строкой, а значение объектом. Gremlin позволяет как проверять наличие свойства с данным ключом у данной вершины или ребра, так и получать значение свойства, а также использовать это значение в самом запросе, например для фильтрации. С примером запроса, зависящего от свойств, можно ознакомиться на листинге 2.

Листинг 2 – Пример запроса на Gremlin, зависящего от свойств

```
g.V().where(values('terminals').is(gt(3)))
```

TinkerPop, в свою очередь, предоставляет интерфейс `Property`, который используется для получения доступа к свойствам данной вершины или ребра. Имеется возможность как получить доступ ко всем доступным свойствам вершины или ребра, так и к конкретным свойствам, предоставив список ключей этих свойств. Этот список будет являться фильтром для доступных у данной вершины или ребра свойств.

Помимо свойств, TinkerPop также вводит понятие «меток». Вершины и ребра могут обладать единственной строковой меткой. Так как метка единственна, для получения доступа к ней не требуется предоставлять ключ. С диаграммой модели свойств в TinkerPop можно ознакомиться на рисунке 2.

2.3.2. Свойства в WebGraph

Эти метки можно считать свойством ребра из изначальной вершины в соседа. Для каждого ребра можно хранить единственный объект метку. В осталь-

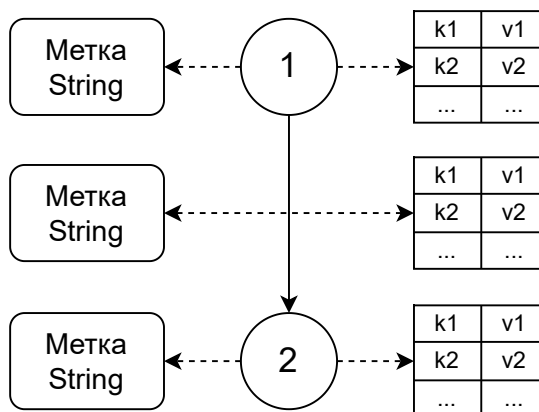


Рисунок 2 – Архитектура свойств в TinkerPop

ном работа со свойствами перекладывается на пользователя, в частности свойства вершин или хранение нескольких свойств для ребер. Диаграмма модели свойств в WebGraph представлена на рисунке 3.

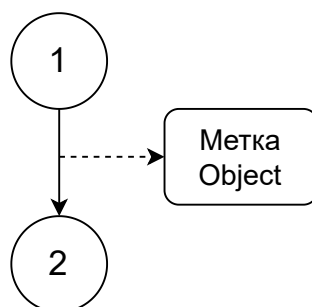


Рисунок 3 – Архитектура свойств в WebGraph

Тем не менее хранение свойств вершин и ребер, помимо самой структуры графа, является важной элементом работы с графом, ведь как обсуждалось в разделе 2.3, запросы зачастую представляют интерес именно в привязке к свойствам. Поэтому существует ряд стандартных практик, использующихся для хранения и получения доступа к свойствам при работе с WebGraph. К примеру для свойств вершин может использоваться сериализация массива примитивных значений, в котором индекс в массиве соответствует индексу вершины, а значение по индексу — значению свойства для соответствующей вершины. Следует реализовать ряд классов, позволяющих пользователю удобным образом задавать собственные способы хранения свойств, а также использовать стандартные.

2.4. Анализ производительности реализации

TinkerPop предоставляет широкие возможности для обхода графов, а также в основном используется в графовых базах данных. WebGraph, в свою

очередь фокусируется на достижении малого отпечатка в памяти, и используется в том случае, когда эффективность и минимальные накладные расходы критически необходимы. При связке этих двух фреймворков могут возникнуть проблемы с производительностью — требуется как сохранить малый отпечаток в памяти, достигаемый сжатием графа, так и оценить, не становятся ли критическими накладные расходы, которые появляются при адаптации WebGraph к TinkerPop.

Чтобы провести анализ производительности, предлагается выбрать конкретный набор данных, представленный в виде графа. Следует определить ряд запросов, представляющих на нем интерес, после чего реализовать данные запросы на языке Gremlin, и, используя сжатое WebGraph представление данного графа и разработанную связку, исполнить данные запросы, проанализировав результаты по ряду метрик, таких как:

- время выполнения;
- используемая память;
- число вершин и ребер, затронутых в процессе исполнения;
- число результатов;

Подобный анализ позволит оценить потерю в эффективности, вызванную разработанной связкой, и сделать выводы о практической применимости решения.

Выводы по главе 2

В этой главе были подробнее рассмотрены конкретные шаги, необходимые для достижения поставленной задачи. Описаны основные интерфейсы WebGraph и TinkerPop, отличия в моделях свойств. Кроме того был описан подход к верификации решения.

ГЛАВА 3. РЕАЛИЗАЦИЯ БИБЛИОТЕКИ, СВЯЗЫВАЮЩЕЙ TINKERPOP И WEBGRAPH

Фреймворк WebGraph реализован на языке Java, и так как решение нацелено на случай, в котором пользователь уже использует WebGraph, и желает получить возможность исполнять на доступном графе Gremlin запросы, решение также было реализовано на языке Java. TinkerPop также реализован на языке Java, что упрощает разработку взаимодействия двух фреймворков. В этой главе будут подробнее описаны детали реализации.

3.1. Реализация TinkerPop

В разделе 2.2.2 были описаны интерфейсы и методы, предоставляемые TinkerPop, реализация которых позволяет исполнять в ней Gremlin запросы (англ. TinkerPop-enabled system). Согласно конвенции реализации этих интерфейсов должны именоваться из двух частей: название системы и название интерфейса в качестве суффикса. В качестве названия системы естественным образом было решено использовать WebGraph. Далее эти интерфейсы и их реализации будут рассмотрены подробнее.

3.1.1. WebGraphGraph

Интерфейс Graph является основной точкой входа TinkerPop. Согласно конвенции его реализация для определенной системы должна содержать публичный статический метод (или несколько методов), возвращающий экземпляр этого класса, и этот метод должен использоваться для создания объектов интерфейса Graph. Для взаимодействия с WebGraph этот метод должен получать в качестве параметра класс, предоставляющий доступ к сжатому графу. На момент начала работы единственным подходящим классом был ImmutableGraph, позволяющий обходить граф в одном направлении. Поскольку TinkerPop позволяет обходить граф как в прямом, так и в обратном направлении ребер, после консультации с командой WebGraph, во фреймворк был добавлен новый класс BidirectionalImmutableGraph [15], позволяющий получать не только соседей по направлению ребер, но и соседей в обратном направлении. Именно этот класс будет использоваться как основная точка доступа к структуре сжатого графа, а методы TinkerPop будут делегироваться этому классу.

Помимо создания графа интерфейс `Graph` позволяет получать доступ к вершинам и ребрам графа по их идентификаторам, а также ко всем вершинам или ребрам.

3.1.1.1. Получение вершин

Доступ к вершинам в `TinkerPop` осуществляется через метод `vertices(Object... vertexIds)`, возвращающий итератор по вершинам с данными индексами, либо итератор по всем вершинам, если массив предоставленных индексов пустой. Так как индексы вершин в `WebGraph` это числа типа `long`, ожидается, что будут предоставлены идентификаторы именно этого типа. Для предоставленных индексов возвращаются объекты-обертки над ними класса `WebGraphVertex`, описанного в разделе 3.1.2. Если массив индексов пустой, обертки возвращаются над индексами, полученными методом `nodeIterator()` класса `ImmutableGraph`, возвращающем итератор по всем вершинам графа.

3.1.1.2. Получение ребер

Доступ к ребрам в `TinkerPop` осуществляется аналогично вершинам — метод `edges(Object... edgeIds)` возвращает итератор по ребрам с данными индексами, либо итератор по всем ребрам, если массив индексов пустой. Так как абстракции над ребрами в `WebGraph` нет, идентификаторы ребер должны быть синтетическими. Поскольку `WebGraph` не поддерживает кратные ребра, в качестве такого идентификатора можно использовать пару индексов, обозначающих исходящую и входящую вершины. Для этого использовался класс `LongLongPair` библиотеки `fastutil` [16]. Для предоставленных индексов возвращаются объекты-обертки над ними класса `WebGraphEdge`, описанного в разделе 3.1.3. Если массив индексов пустой, обертки возвращаются над индексами, в которых исходящая вершина получается методом `nodeIterator()` класса `ImmutableGraph`, а входящая вершина методом `successors(long vertexId)` со значением аргумента, равным индексу исходящей вершины, возвращающем итератор по индексам соседей исходящей вершины. Таким образом перебираются все начальные вершины, а для каждой начальной вершины все конечные. Все пары таких вершин описывают все ребра графа.

3.1.2. WebGraphVertex

Интерфейс `Vertex` обозначает вершину графа. Он предоставляет доступ к идентификатору вершины, ее метке, лежащему в основе графу (эти три метода выделены в интерфейс `Element`), входящим и исходящим вершинам и ребрам, а также свойствам вершины. Идентификатор и граф сопровождаются в конструкторе `WebGraphVertex`, о метках и свойствах речь пойдет в разделах 3.1.4 и 3.2. Для получения вершин и ребер используются методы `successors(long vertexId)` и `predecessors(long vertexId)` класса `BidirectionalImmutableGraph`. Возвращаемый ими `LazyLongIterator` оборачивается в стандартный `Iterator` пакета `java.util`. При доступе к вершинам возвращаются обертки над индексом вершины-соседа, а при доступе к ребрам — обертки над парой индексов: текущей вершины и полученного соседа.

3.1.3. WebGraphEdge

Интерфейс `Edge` обозначает ребро графа. Он предоставляет доступ к описанным выше методам интерфейса `Element` (идентификатор, метка, граф), входящей и исходящей вершинам, а также свойствам ребра. Идентификатор и граф сопровождаются в конструкторе `WebGraphEdge`, о метках и свойствах речь пойдет в разделах 3.1.4 и 3.2. Для входящей и исходящей вершин возвращаются обертки над их индексами.

3.1.4. WebGraphProperty

Доступ к свойствам в `TinkerPop` осуществляется через метод `properties(String propertyKeys)` интерфейса `Element`, общего у `Vertex` и `Edge`, где `propertyKeys` являются фильтром над доступными свойствами. Если ключей предоставлено не было, возвращаются все доступные свойства.

Сам интерфейс `Property` является аналогом `Optional` пакета `java.util`, помимо значения обладающим дополнительным строковым ключом. Определяет методы для получения ключа, значения, проверки наличия значения. При этом как упоминалось в разделе 2.2.2, `TinkerPop` поддерживает метасвойства вершин, поэтому для свойств вершин используется отдельный интерфейс `VertexProperty`, объединяющий интерфейсы `Property` и `Element`. В рамках работы добавление поддержки метасвойств не планируется, поэтому методы получения свойств для

класса `WebGraphVertexProperty` вызывают стандартное исключение `VertexProperty.Exceptions.metaPropertiesNotSupported()`.

В классах `WebGraphVertex` и `WebGraphEdge` реализация метода `properties(String propertyKeys)` получает доступные свойства и их значения через класс `WebGraphPropertyProvider`, речь о котором пойдет в разделе 3.2.1. Пользуясь либо предоставленными ключами, либо списком всех доступных ключей, возвращается итератор по оберткам свойств с отфильтрованными `null` значениями.

3.2. Реализация работы со свойствами

Модели свойств в `TinkerPop` и `WebGraph` заметно отличаются. Более того, поддержка свойств при работе с `WebGraph` во многом ложится на самого пользователя. Чтобы наладить взаимодействие между этими моделями свойств, а также упростить работу со стандартными подходами к хранению свойств в `WebGraph`, был разработан ряд классов, речь о которых пойдет далее. Схема архитектуры свойств представлена на рисунке 4.

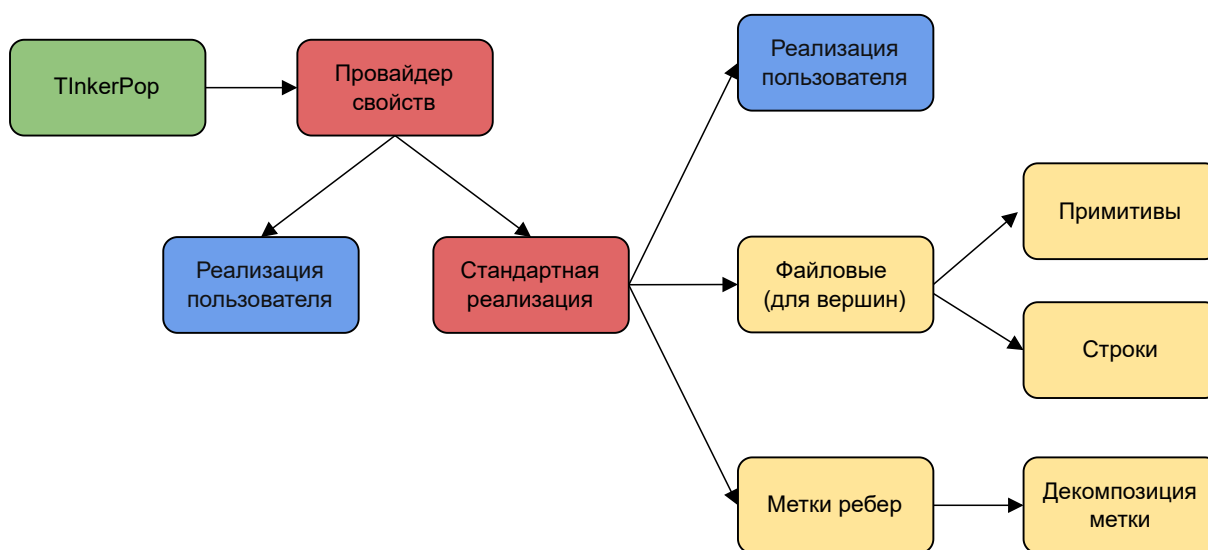


Рисунок 4 – Общая схема архитектуры свойств

3.2.1. Провайдер свойств

Интерфейс `WebGraphPropertyProvider` отвечает за доступ к меткам вершин и ребер, доступным ключам свойств, а также к значениям этих свойств. Он является связующим звеном между моделью свойств в `TinkerPop` и работой со свойствами в `WebGraph`. Экземпляр этого интерфейса наравне с экземпляром `BidirectionalImmutableGraph` необходим для создания `WebGraphGraph`.

3.2.2. Стандартный провайдер свойств

Пользователь может как предоставить свою реализацию `WebGraphPropertyProvider`, так и использовать стандартную `StandardWebGraphPropertyProvider`. Стандартная реализация позволяет определять метки вершин и ребер, предоставив функцию из индекса в строковую метку, а также регистрировать свойства при помощи отдельных классов `VertexProperty` и `EdgeProperty`. Конструкторы этих классов принимают строковый ключ свойства, а также функцию, возвращающую значение по идентификатору вершины или ребра. Эти свойства регистрируются в провайдере. Метод, возвращающий доступные ключи, возвращает ключи всех зарегистрированных свойств. По ключу свойства и идентификатору вершины или ребра провайдер обращается к соответствующему зарегистрированному свойству и возвращает полученное значение. Провайдер имеет доступ ко всем зарегистрированным свойствам, поэтому при попытке регистрации свойства с занятым ключом, пользователь получает сообщение о том, что свойство с данным ключом уже существует.

3.2.3. Стандартные свойства вершин

Как упоминалось в разделе 2.3.2, `WebGraph` не предоставляет стандартных методов работы со свойствами вершин — эта задача перекладывается на пользователя. При этом пользователи зачастую используют ряд стандартных практик для работы с ними. Чтобы упростить работу со свойствами в разрабатываемой связке, часть таких стандартных практик была реализована. Пользователь имеет возможность определять свои `VertexProperty`, однако если он решает воспользоваться одним из стандартных методов, от него потребуется минимальная настройка. Далее будут подробнее разобраны стандартные подходы и их реализации.

3.2.3.1. Файловые свойства вершин

Поскольку `WebGraph` не поддерживает свойства вершин, чаще всего пользователи сохраняют их в отдельных файлах вместе со структурой графа. Удобным способом хранения свойств вершин является хранение массива значений в сериализованном виде. Индекс в массиве соответствует индексу вершины, а значение по индексу — значению свойства для соответствующей вершины. Для сохранения и чтения массива хорошо подходит библиотека

`fastutil` от создателей `WebGraph`. Она уже используется в самом фреймворке `WebGraph` и ее использование напрашивается и для работы со свойствами. Эта библиотека позволяет сериализовывать и читать из файла массивы примитивов, что дает возможность определить файловые свойства вершин для всех примитивов. Для этого был создан класс `FileVertexProperty`, принимающий ключ свойства, тип значения и путь к файлу с массивом сериализованных значений. По типу свойства класс выбирает нужный способ чтения файла и позволяет получать доступ к значениям.

Помимо примитивов, важным типом свойств является строковый тип, так как многие данные для вершин и ребер имеют именно текстовое представление. Для того чтобы добавить поддержку строковых свойств вершин и ребер, был реализован следующий метод хранения. В первом файле хранится буфер — байтовый массив блоков, каждый из которых соответствует отдельной строке. Каждый блок состоит из двух частей: первые 4 байта хранят число n — длину строки, а последующие n байт — саму строку в байтовом представлении. Во втором массиве типа `long` хранятся сдвиги для каждой из вершин — указатель на индекс начала нужного блока в буфере. Чтобы получить значение строки для данной вершины, следует сначала получить соответствующий вершине сдвиг, затем по данному сдвигу прочитать первые 4 байта блока, получив длину строки n , а затем считать последующие n байт блока — байтовый массив самой строки. Воспользовавшись конструктором `String(byte[] bytes)` можно получить связанную с данной вершиной строку. С диаграммой данного метода можно ознакомиться на рисунке 5. Описанный механизм реализован в классе `StringFileVertexPropertyGetter`, конструктор которого принимает пути к файлам с буфером и сдвигами и позволяет получить доступ к значениям.

3.2.4. Стандартные свойства ребер

Как обсуждалось в разделе 2.3.2, `WebGraph` поддерживает один способ указания свойств ребер — метки ребер. Помимо метода `sucessors(long vertexId)`, возвращающего итератор по соседям вершины, есть также метод `labelledSucessors(long vertexId)`, доступный у класса `ArcLabelledImmutableGraph` и возвращающий итератор по помеченным соседям. Таким образом метка соседа является меткой ребра из изначальной вершины в соседа. Каждое ребро может обладать одной меткой типа

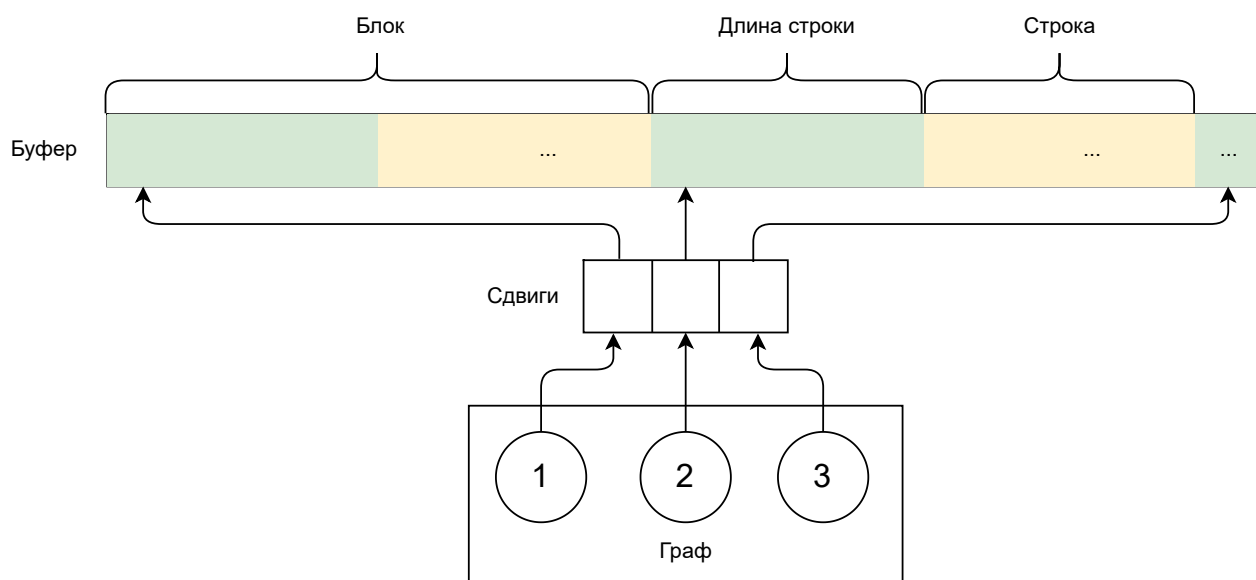


Рисунок 5 – Хранение строкового свойства вершины

Object. В разработанной библиотеке пользователь имеет возможность определять свои `EdgeProperty`, однако для меток ребер был разработан отдельный класс, упрощающий работу с ними.

3.2.4.1. `ArcLabelEdgeProperty`

Для того чтобы предоставить `TinkerPop` доступ к меткам ребер, был разработан класс `ArcLabelEdgeProperty`. В конструкторе он принимает экземпляр помеченного графа `ArcLabelledImmutableGraph`, к которому должен иметь доступ пользователь, если он работает с метками в `WebGraph`. Чтобы получить метку ребра из вершины a в вершину b берется итератор по помеченным соседям вершины a , находится ребро в вершину b и берется его метка. Поскольку подобное свойство у графа может быть только одно, оно обладает зарезервированным ключом. Это предотвращает возможность зарегистрировать одно и то же свойство под разными ключами.

Поскольку метка единственна для ребра, зачастую хранение нескольких свойств реализуется за счет хранения нескольких полей в объекте метки, где каждое поле отвечает за отдельное свойство. Для того чтобы предоставить возможность декомпонировать метку на несколько свойств с различными ключами, минуя необходимость декомпонировать метку самостоятельно в каждом `Gremlin` запросе, был добавлен класс `ArcLabelEdgeSubProperty`, принимающий в конструкторе ключ свойства и функцию, отбирающую требуемое значение из полной метки. В реализации это свойство обращается к полной

метке и возвращает результат применения функции к объекту метки. С примером использования декомпозиции метки можно ознакомиться на листинге 3. Помимо декомпозиции метки этот класс также дает возможность создавать синтетические свойства, то есть произвольные трансформации над изначальной меткой. К примеру таким образом можно объединить два поля в одно отдельное свойство.

Листинг 3 – Пример декомпозиции метки

```
var arcLabel = new ArcLabelEdgeProperty<MyArcLabel>(myGraph.
    getLabelledGraph());
p.addEdgeProperty(new ArcLabelEdgeSubProperty<>("duration",
    arcLabel, MyArcLabel::getDuration));
```

3.3. Исполнитель запросов

Для того чтобы предоставить пользователю возможность запускать Gremlin запросы на графе, был разработан класс `GremlinQueryExecutor`, принимающий в конструкторе экземпляр интерфейса `Graph` (подразумевается, что это будет объект класса `WebGraphGraph`), и предоставляющий набор методов для запуска запросов на этом графе. Базовым методом является метод `eval(String query)` (реализация представлена в приложении А.1), делегирующий исполнение запроса классу `GremlinExecutor`, предоставляемому `TinkerPop`.

Для удобства работы с результатами запросов были добавлены два вспомогательных метода: `print(String query)` выводит в консоль все результаты запроса по мере исполнения, `get(String query)` возвращает список результатов для дальнейшего использования в программе.

Для профилирования запросов Gremlin предоставляет стандартный шаг `profile()`. Для удобства профилирования в исполнитель запросов был добавлен метод `profile(String query)`, добавляющий этот шаг к изначальному запросу. Помимо этого, для работы с запросами, исполнение которых занимает заметное время, был добавлен метод `time(String query, long total)`, принимающий запрос и ожидаемое число результатов, и по ходу исполнения запроса выводящий в консоль статистику: число обработанных результатов, скорость обработки и прогноз оставшегося времени исполнения.

Gremlin предоставляет возможность формулировать запросы не только в качестве строк, но и при помощи классов и методов (англ. Gremlin-Java DSL). Это позволяет проводить статистический анализ запросов, в том числе типов, делает возможным автодополнение и использование javadoc для шагов, что упрощает формулировку запросов. Запрос, сформулированный таким образом, представлен классом `GraphTraversal`, а стартовой точкой является класс `GraphTraversalSource`. Все описанные выше методы класса `GremlinQueryExecutor` имеют аналог, принимающий в качестве аргумента не строковый запрос, а функцию из `GraphTraversalSource` в `GraphTraversal`, тем самым позволяя запускать запросы, сформулированные на Gremlin-Java DSL. С примерами запуска идентичных запросов, сформулированных с применением двух описанными подходов, можно ознакомиться на листингах 4 и 5.

Листинг 4 – Строковый Gremlin запрос

```
executor.print("g.V().not(in())");
```

Листинг 5 – Запрос на Gremlin-Java DSL

```
executor.print(g -> g.V().not(in()));
```

3.4. Использование разработанной библиотеки

Решение было реализовано в качестве Java библиотеки `webgraph-tinkerpop`. Пользователь, уже использующий `WebGraph` в своей системе, имеет возможность добавить разработанную библиотеку в зависимости, после чего он получает доступ к конструктору `WebGraphGraph`, классам для работы со свойствами, и исполнителю Gremlin запросов `GremlinQueryExecutor`.

Библиотека опубликована на хостинге GitHub и доступна по ссылке <https://github.com/andrey-star/webgraph-tinkerpop>. Библиотека не опубликована в центральном репозитории Maven [17], поэтому для добавления в зависимости своего проекта требуется скачать репозиторий и установить библиотеку в локальный Maven репозиторий при помощи команды `mvn install`, вызванной в корневой директории скачанного проекта.

Для добавления самой зависимости требуется отредактировать файл `pom.xml` своего проекта и добавить в раздел `dependencies` строки, приведенные на листинге 6.

Листинг 6 – Добавление `webgraph-tinkerpop` в зависимости проекта

```
<dependency>
  <groupId>org.webgraph.tinkerpop</groupId>
  <artifactId>webgraph-tinkerpop</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

С примером использования библиотеки для регистрации свойств и исполнения Gremlin запросов на графе можно ознакомиться на листинге 7. Здесь создается стандартный провайдер свойств, добавляется метка вершины, поставляемая классом графа, файловое свойство вершины `timestamp`, а также свойство вершины `name`, поставляемое графом. Для ребер создается стандартная метка ребра, а также добавляется декомпозиция этого свойства для поля `duration`. После этого создается экземпляр класса `WebGraphGraph` при помощи метода `open`, принимающего изначальный граф и провайдер свойств. При помощи исполнителя запросов на этом графе запускаются два запроса, один в строковом представлении, другой на Gremlin-Java DSL.

3.5. Внедрение результатов в проект Software Heritage

Библиотека была использована для того, чтобы добавить возможность запускать Gremlin запросы на графе Software Heritage. Для этого в репозиторий `swh-graph` [18] была добавлена зависимость на разработанную библиотеку `webgraph-tinkerpop`.

Сервер Software Heritage уже имеет доступ к `BidirectionalImmutableGraph`, дополнительно обернутый в класс `SwhBidirectionalGraph`, поэтому для успешного использования библиотеки на том же сервере остается настроить использование свойств и получить экземпляр `WebGraphGraph` при помощи статического метода, описанного выше.

Для настройки свойств использовался класс `StandardWebGraphPropertyProvider`. В качестве метки вершины использовался тип вершины, получаемый методом `getNodeType(long vertexId)` класса `SwhBidirectionalGraph`. Меток ребер (имеются

Листинг 7 – Пример использования библиотеки для регистрации свойств и исполнения Gremlin запросов

```
var p = new StandardWebGraphPropertyProvider();
p.setVertexLabeller(myGraph::getNodeTypes);
p.addVertexProperty(new FileVertexProperty<>("timestamp", Long.
    class, Path.of("timestamp.bin")));
p.addVertexProperty(new VertexProperty<>("name", myGraph::getName)
    );

var arcLabel = new ArcLabelEdgeProperty<MyArcLabel>(myGraph.
    getLabelledGraph());
p.addEdgeProperty(arcLabel);
p.addEdgeProperty(new ArcLabelEdgeSubProperty<>("duration",
    arcLabel, MyArcLabel::getDuration));

try (var gremlinGraph = WebGraphGraph.open(myGraph, p, myGraph.
    getPath())) {
    GremlinQueryExecutor executor = new GremlinQueryExecutor(
        gremlinGraph);
    executor.print(g -> g.V().has("timestamp", gt(1000)));
    executor.print("g.V().hasLabel('REV')");
}
```

в виду строковые метки TinkerPop, не метки ребер WebGraph) рассматриваемый домен не предусматривает. Для добавления свойства вершины `author_timestamp` использовался класс `FileVertexProperty`. Также были добавлены два свойства вершины, не подходящие под стандартные техники, и напрямую предоставляемые классом `SwhBidirectionalGraph`. Для этого использовался класс `VertexProperty`. Для добавления свойства ребра был использован класс `ArcLabelEdgeProperty`, не требующий никаких настроек, достаточно предоставить `ArcLabelledImmutableGraph`. Также было добавлено дополнительное свойство поверх свойства ребра, конвертирующее набор файлов, связанных с ребром, в пары из строкового представления файла и числовой репрезентации прав доступа. Добавление этих свойств приведено в приложении Б.1.

Выводы по главе 3

В третьей главе были подробно рассмотрены детали реализации TinkerPop для WebGraph, а именно исполнитель Gremlin запросов, взаимодействие структурных методов, а также моделей свойств двух фреймворков. Были описаны подходы к работе со свойствами в WebGraph и рассказано, каким об-

разом эти методы были реализованы в библиотеке. В конце был приведен пример практического использования библиотеки командой Software Heritage.

ГЛАВА 4. СРАВНЕНИЕ РАЗРАБОТАННОЙ БИБЛИОТЕКИ И НАТИВНОГО ПОДХОДА

Разработанная библиотека предоставляет возможность пользоваться преимуществами Gremlin в формулировке запросов, при этом негативно влияя на эффективность их исполнения по сравнению с нативными реализациями запросов. Для оценки эффективности был выбран реальный набор данных, в нем сформулированы представляющие интерес запросы. Эти запросы были реализованы на Gremlin, а также в нативном виде, после чего было проведено сравнение исполнения запросов по различным метрикам. В этой главе будет подробнее описан процесс тестирования и представлены его результаты, а также описаны получаемые пользователем преимущества использования Gremlin перед нативным подходом к реализации запросов.

4.1. Набор данных

В качестве домена был выбран архив Software Heritage, содержащий данные систем контроля версий репозитория с открытым исходным кодом. Единый граф Software Heritage содержит более 20 миллиардов вершин и более 230 миллиардов ребер. Со структурой графа можно ознакомиться на рисунке 1. Для тестов был выбран набор данных `python3k` [13], являющийся частью основного, и содержащий данные о трех тысячах популярных репозиториях на языке Python. Граф содержит 46 миллионов вершин и 1 миллиард 218 миллионов ребер. Software Heritage использует WebGraph для сжатия структуры графа, поэтому набор данных содержит все необходимые для работы с WebGraph файлы.

4.2. Формулировка запросов

Были сформулированы три запроса, представляющие интерес в данном домене:

- самая ранняя ревизия для файла или директории;
- список файлов ревизии;
- дерево ревизий снимка;

4.2.1. Самая ранняя ревизия для файла или директории

Принимая в качестве аргумента индекс файла или директории, этот запрос находит все ревизии, достижимые из этого файла или директории, и возвращает минимальную ревизию по значению свойства `author_timestamp`.

Таким образом результатом запроса является первая ревизия, в которой данный файл или директория появились в том виде, в котором они представлены в вершине, являющейся аргументом запроса. Область запроса представлена на рисунке 6.



Рисунок 6 – Область графа, затрагиваемая запросом «Первая ревизия файла или директории»

4.2.2. Список файлов ревизии

Принимая на вход в качестве аргумента индекс ревизии или директории, этот запрос возвращает пути и права доступа для всех доступных файлов и директорий данной ревизии или директории. Таким образом этот запрос является аналогом команды `ls -lR`. Названия директорий и файлов хранятся в метке ребра. Область запроса представлена на рисунке 7.

4.2.3. Дерево ревизий снимка

Принимая на вход в качестве аргумента индекс снимка, этот запрос возвращает все связи между снимками, ревизиями и релизами. При этом, если ребро следует из снимка в ревизию, то эта связь представляет ветку, и результат сопровождается ее названием. Если ребро следует из снимка в релиз, то результат сопровождается названием релиза. Названия веток и релизов хранятся в метке ребра. Область запроса представлена на рисунке 8.



Рисунок 7 – Область графа, затрагиваемая запросом «Список файлов ревизии»

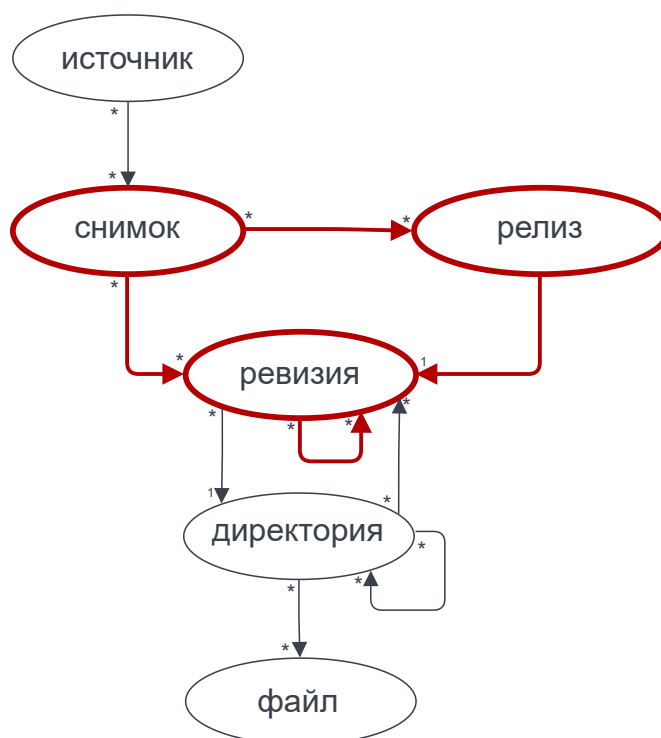


Рисунок 8 – Область графа, затрагиваемая запросом «Дерево ревизий снимка»

4.3. Реализация запросов на Gremlin

Для того чтобы протестировать разработанную библиотеку на описанных запросах, требуется предварительно реализовать их на Gremlin. Запросы

на Gremlin представляют собой цепочки «шагов». Каждый шаг либо оперирует результатами, полученными на предыдущем шаге, либо совершает стороннее действие (англ. side effect). Также существует два стартовых шага `V()` и `E()`, возвращающих вершины и ребра графа соответственно, опционально принимая идентификаторы интересующих вершин или ребер. Подробнее с доступными шагами и их действиями можно ознакомиться в документации к Gremlin [19]. Далее будут представлены и прокомментированы реализации трех описанных запросов.

4.3.1. Самая ранняя ревизия для файла или директории

Реализация запроса представлена на листинге 8.

- а) начинаем в файле или директории;
- б) рекурсивно переходим в предка, пропуская посещенные вершины;
- в) сохраняем все встреченные ревизии;
- г) избавляемся от повторений;
- д) сортируем по значению свойства `author_timestamp` и берем первый результат;

Листинг 8 – Самая ранняя ревизия для файла или директории на Gremlin

```
g.V(v) // а
  .repeat(in().dedup()) // б
  .emit(hasLabel("REV")) // в
  .dedup() // г
  .order().by("author_timestamp", Order.asc).limit(1) // д
```

4.3.2. Список файлов ревизии

Реализация запроса представлена на листинге 9.

- а) начинаем в ревизии или директории;
- б) если стартовая вершина это ревизия, переходим в соответствующую директорию;
- в) рекурсивно переходим в потомка, сохраняя пройденное ребро;
- г) для подмодулей переходим в соответствующую директорию;
- д) возвращаем каждую пройденную вершину;
- е) для каждой вершины рассматриваем весь пройденный до нее путь;
- ж) для ребер на пути возвращаем свойство `dir_entry_str`;
- и) конвертируем путь в графе в файловый путь;

- к) вычисляем путь в родительскую директорию текущей вершины;
- л) если последнее ребро одно, добавляем название к пути;
- м) если последних ребер несколько, возвращаем итератор по всем путям;

Листинг 9 – Список файлов ревизии на Gremlin

```

g.V(root) // а
  .choose(hasLabel("REV"), out().hasLabel("DIR")) // б
  .repeat(outE().inV() // в
    .choose(hasLabel("REV"), out().hasLabel("DIR"))) // г
  .emit() // д
  .path() // е
  .map(unfold()
    .<DirEntryString[]>values("dir_entry_str") // ж
    .fold()
  ).flatMap(path -> { // и
    List<DirEntryString[]> pathDirs = path.get();
    StringBuilder dir = new StringBuilder();
    for (int i = 0; i < pathDirs.size() - 1; i++) { // к
      dir.append(pathDirs.get(i)[0].filename).append("/");
    }
    DirEntryString[] last = pathDirs.get(pathDirs.size() - 1);
    if (last.length == 1) { // л
      var entry = last[0];
      return List.of(String.format("%s%s [perms: %s]", dir,
        entry.filename, entry.permission)).iterator();
    }
    List<String> res = new ArrayList<>();
    for (DirEntryString entry : last) { // м
      res.add(String.format("%s%s [perms: %s]", dir, entry.
        filename, entry.permission));
    }
    return res.iterator();
  })

```

4.3.3. Дерево ревизий снимка

Реализация запроса представлена на листинге 10.

- а) рекурсивно переходим в потомка, пропуская посещенные ребра;
- б) оставляем только ребра, оканчивающиеся ревизией или снимком;
- в) сохраняем оставшиеся ребра;
- г) переходим в оставшиеся ребра, избегая повторений;
- д) возвращаем множество встреченных ребер;
- е) конвертируем список ребер в индивидуальные ребра;
- ж) получаем значение свойства, хранящего названия веток или релизов;
- и) конвертируем ребра во все доступные ветки или релизы;

- к) получаем ребро, далее извлекаем идентификаторы и метки вершин;
- л) если изначальная вершина это снимок, добавляем информацию о ветке или релизе;

Листинг 10 – Дерево ревизий снимка на Gremlin

```

g.withSideEffect("e", new HashSet<>())
.V(snapshot)
.repeat(outE().where(P.without("e")) // а
      .where(inV().hasLabel("REV", "REL")) // б
      .aggregate("e") // в
      .inV().dedup() // г
.<Edge>cap("e") // д
.unfold() // е
.elementMap("filenames") // ж
.flatMap(edgeElementMapTraverser -> { // и
    Map<Object, Object> edgeElementMap = edgeElementMapTraverser.
        get(); // к
    long outId = (long) ((Map<Object, Object>) edgeElementMap.get(
        Direction.OUT)).get(T.id);
    long inId = (long) ((Map<Object, Object>) edgeElementMap.get(
        Direction.IN)).get(T.id);
    String outLabel = (String) ((Map<Object, Object>)
        edgeElementMap.get(Direction.OUT)).get(T.label);

    String edgeStr = String.format("(%s -> %s)", outId, inId);
    if (outLabel.equals("SNP")) { // л
        String[] branches = (String[]) edgeElementMap.get("
            filenames");
        List<String> res = new ArrayList<>(branches.length);
        for (String branch : branches) {
            res.add(edgeStr + " " + branch);
        }
        return res.iterator();
    }
    return List.of(edgeStr).iterator();
})

```

4.4. Проведение тестирования

Для измерения производительности запросов был разработан класс `Benchmark`, запускающий и профилирующий описанные запросы на случайном наборе подходящих вершин.

Для этого был создан интерфейс `BenchmarkQuery`, определяющий ряд методов, необходимых профайлеру для измерения показателей запроса, а именно:

- `getName()` — возвращает уникальный строковый идентификатор этого запроса;
- `getQuery()` — возвращает запрос, реализованный на Gremlin-Java DSL;
- `generateStartingPoints()` — возвращает случайный набор подходящих стартовых вершин для данного запроса;
- `nativeImpl(long id)` — нативная реализация этого запроса;

Этот интерфейс был реализован для каждого из трех запросов. Поскольку все три запроса имеют стартовой точкой вершину с определенной меткой, генерация случайных стартовых вершин осуществляется через предварительный Gremlin запрос, находящий вершины с этой меткой, а затем выбирающий случайный набор среди этих вершин. С реализацией этого запроса можно ознакомиться на листинге 11.

Листинг 11 – Генерация случайных стартовых вершин

```
private List<Long> randomVerticesWithLabel(String label, long
    count) {
    return e.get(g -> g.V().hasLabel(label)
        .order().by(Order.shuffle)
        .limit(count)
        .id().map(id -> (long) id.get()));
}
```

Для анализа производительности профайлер генерирует стартовые точки для выбранного запроса, запускает запрос на этих стартовых точках и подводит статистику по мере исполнения. Так как время исполнения одного запроса может варьироваться, каждый запрос исполняется определенное число раз, а результат усредняется. Также исполняется и нативная версия запроса для проведения последующего сравнения.

Класс `Benchmark` принимает в качестве аргументов командной строки ряд параметров, позволяющих настраивать профайлер:

- `graphPath` — путь к Software Heritage графу, на котором будут исполняться запросы;
- `query` — строковый идентификатор запроса, который следует профилировать;
- `samples` — число сгенерированных стартовых точек, на которых будет запускаться запрос;
- `iters` — число запусков запроса для одной стартовой точки;

- `argument` — при наличии пропускается генерация стартовых точек, вместо этого запрос выполняется для переданного аргумента;
- `print` — при наличии вместо промежуточных результатов профайлинга выводятся результаты запроса;

По мере выполнения профайлер выводит текущий аргумент и итерацию, время выполнения Gremlin запроса и нативного запроса, а также число результатов запроса. Для определения времени выполнения запроса и числа результатов используется описанный в разделе 3.3 шаг `profile()`. Также выводится скорость выполнения исходя из общего времени и числа результатов. Промежуточный вывод профайлера представлен на листинге 12.

Листинг 12 – Пример промежуточного вывода профайлера

```
Running query for id: 45673752 (92/100)
Native time: 506ms
1/3 Finished in: 2428ms. Results: 137874
2/3 Finished in: 2623ms. Results: 137874
3/3 Finished in: 2472ms. Results: 137874
Average for id: 45673752 - 2507ms. Per element: 0.02ms (137874
elements).
```

После окончания выполнения выводится общая статистика, такая как: среднее время выполнения запроса, средняя скорость выполнения относительно числа результатов, максимальное время выполнения и соответствующая стартовая точка, а также среднее время и скорость для нативной версии запросов, и коэффициент замедления Gremlin версии по сравнению с нативной. Текстовое представление результата шага `profile()` сохраняется в текстовые файлы для каждого запроса и итерации, что позволяет подробнее анализировать отдельные случаи. Числовые данные сохраняются в сводную таблицу в формате CSV для возможности дальнейшего анализа и построения графиков. С финальным выводом профайлера можно на листинге 13.

4.5. Результаты тестирования

По результатам тестирования можно сделать выводы о том, какое замедление испытывает запрос по сравнению с его нативной реализацией. Хорошей метрикой для оценки этого показателя является скорость обработки элементов. Под число элементов стоит понимать некую оценку мощности запроса. Для многих запросов подходящим числом элементов может служить число результатов запроса, например для запросов «Список файлов ревизии» и «Дерево

Листинг 13 – Пример финального вывода профайлера

```
Average time: 46ms. Per element: 0.02ms
Max time: 2507ms for id 45673752. Per element: 0.02ms (137874
elements)
Native time: 28ms. Per element: 0.01ms
Slower by: 1.63 times
Results saved at: benchmarks/2022-05-11T12:36:39Z-
snapshotRevisionsWithBranches
```

ревизий снимка». Однако для некоторых запросов такая метрика не подходит, например у запроса «Самая ранняя ревизия для файла или директории» всего один результат. В этом случае можно использовать другое число, отражающее мощность запроса, например число пройденных вершин.

Для наглядного сравнения подходов были построены графики скорости обработки элементов. График для запроса «Самая ранняя ревизия для файла или директории» представлен на рисунке 9, для запроса «Список файлов ревизии» на рисунке 10, для запроса «Дерево ревизий снимка» на рисунке 11. На них отображена зависимость времени исполнения запроса от числа элементов для среднего по трем запускам Gremlin запроса, и нативного запуска запроса. Для удобства помимо графика нативного запуска добавлен график нативного запуска с коэффициентом, максимально приближенный к среднему запуску Gremlin. Это позволяет наглядно отобразить коэффициент замедления.

Из графиков для первого и третьего запросов следует, что Gremlin запросы испытывают замедление в 4–5 раз по сравнению с нативным.

При этом на графике для второго запроса видно, что замедление составляет 9 раз. Причиной такой разницы, вероятно, является тот факт, что первый и третий запросы являются алгоритмическим аналогом своих нативных версий. Второй же запрос с точки зрения заложенного алгоритма несколько отличается от нативной реализации. Проблема заключается в том, что наиболее оптимальным методом для данного запроса является поиск с возвратом (англ. *backtracking*). Именно этот подход был реализован в нативной версии, однако изученные мной Gremlin шаги не позволяют осуществить этот подход. Из-за этого в Gremlin реализации приходится работать с каждым путем от начальной вершины до конечной отдельно. Если изменить нативную реализацию так, чтобы она отражала Gremlin реализацию, то в результатах можно увидеть та-

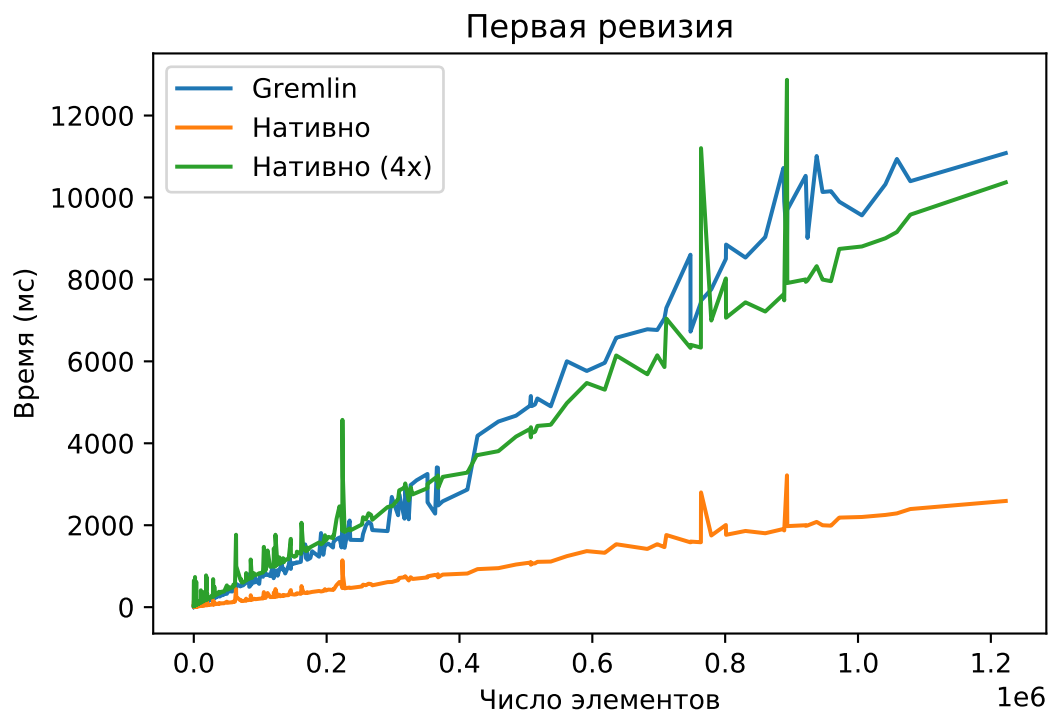


Рисунок 9 – График зависимости времени исполнения запроса от числа элементов для запроса «Самая ранняя ревизия для файла или директории»

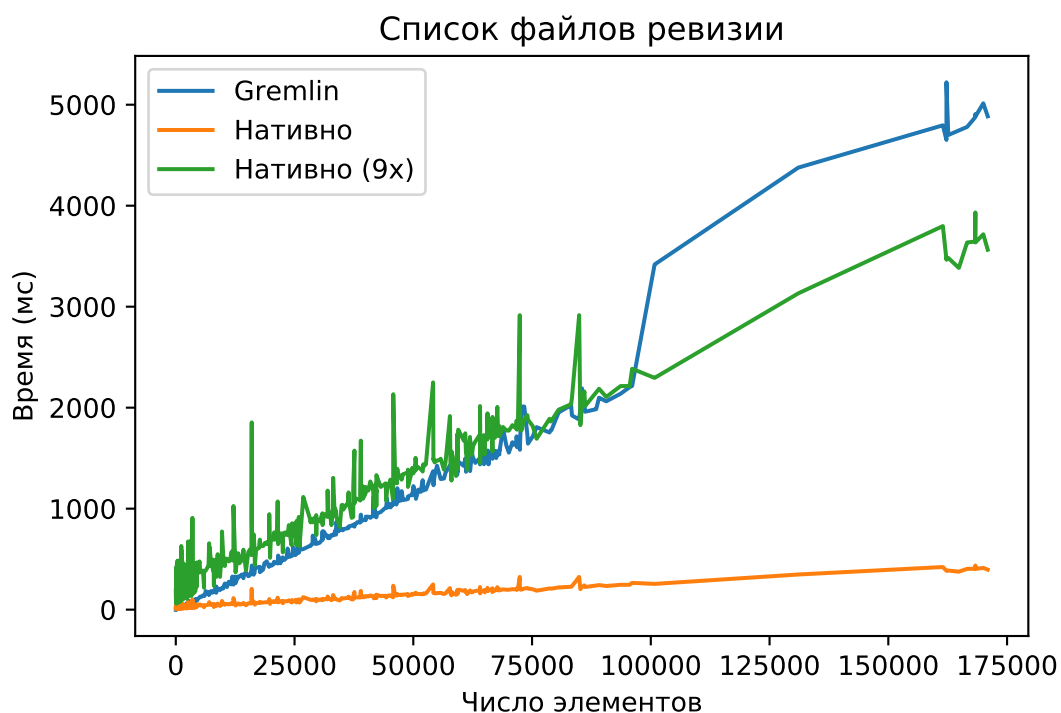


Рисунок 10 – График зависимости времени исполнения запроса от числа элементов для запроса «Список файлов ревизии»

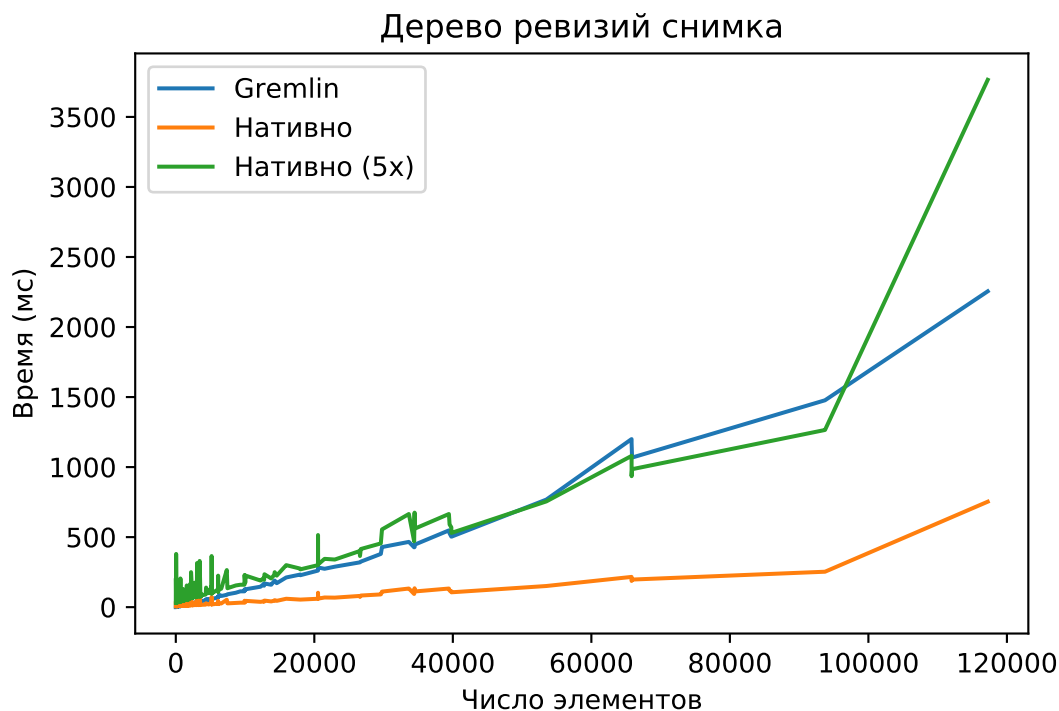


Рисунок 11 – График зависимости времени исполнения запроса от числа элементов для запроса «Дерево ревизий снимка»

кое же замедление в 4–5 раз, как и на других запросах. График для обновленного запроса представлен на рисунке 12.

Из этого можно заключить, что привязка Gremlin к TinkerPop замедляет исполнение запроса в 4–5 раз. При этом важно сформулировать запрос оптимально, что для некоторых специфических случаев не всегда возможно.

4.6. Тестирование на других графах

Чтобы убедиться в том, что полученные результаты тестирования не специфичны для выбранного графа, а универсальны, тесты также были проведены и на нескольких других графах. В интересах полноты тестирования следовало выбрать максимально «отличающиеся» графы. В качестве меры отличия была выбрана средняя степень вершины. Выбрав максимально отличающиеся по средней степени вершины графы, под тестирование попадут графы с весьма разной структурой, что поможет сделать выводы об универсальности полученных ранее результатов.

В качестве тестируемых графов были выбраны два дополнительных набора данных, доступных на странице WebGraph [20], а именно hollywood-2011 и imdb-2021. Среди представленных наборов данных

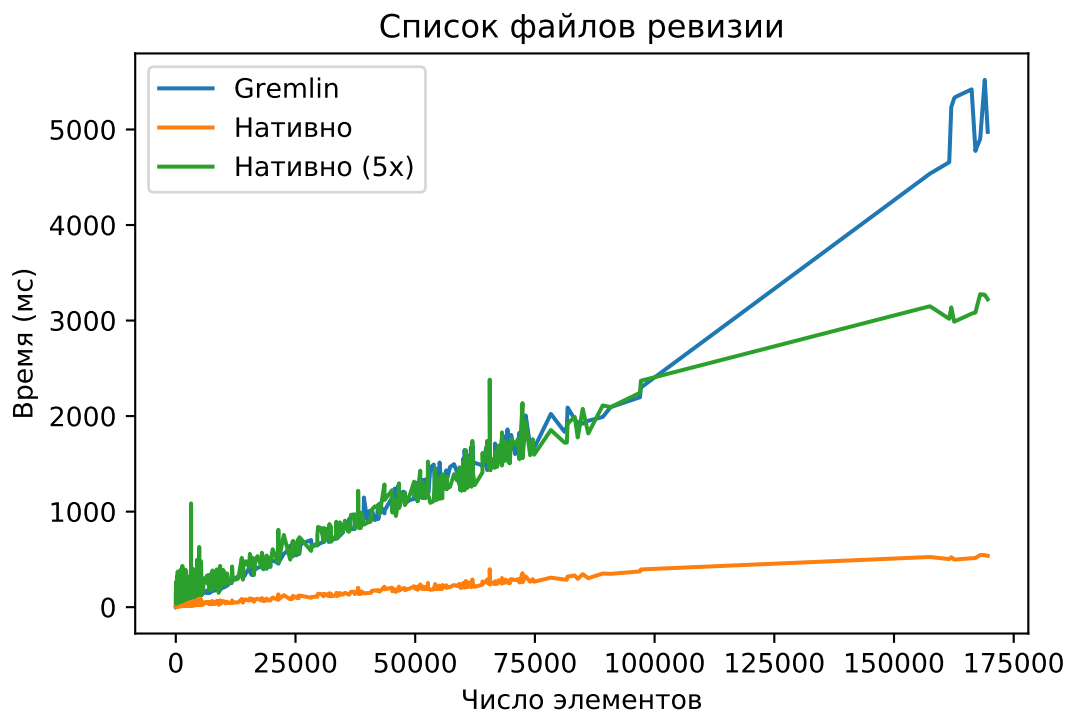


Рисунок 12 – График зависимости времени исполнения запроса от числа элементов для запроса «Список файлов ревизии»

эти наборы максимально отличаются от `python3k` по средней степени вершины: 105,0 у `hollywood-2011` и 3,58 у `imdb-2021`, вместо 26,7 у `python3k`. Формулировка и реализация доменно-специфических запросов на этих графах потребовала бы много времени, поэтому было решено определить один запрос общий для всех трех графов. В качестве такого запроса был выбран обход в глубину. Такой запрос обходит весь граф и не зависит от его свойств. Реализация запроса на Gremlin представлена на листинге 14.

Листинг 14 – Запрос, обходящий весь граф

```
g.V().not(in())
    .repeat(out().dedup())
```

Проведенный анализ времени работы подтвердил полученные ранее результаты: замедление исполнения запроса в 4–5 раз по сравнению с нативной реализацией. Объем дополнительно затрачиваемой памяти сравним с размером помеченного графа в памяти, что означает двукратное ухудшение. При этом стоит отметить, что подобный запрос даже в нативном виде не является осуществимым на больших графах, так как обходит и сохраняет все вершины,

поэтому в реальных случаях ухудшение в использовании памяти будет меньше.

4.7. Преимущества библиотеки перед нативными запросами

Использование Gremlin позволяет существенно сократить размер и сложность запроса, что уменьшает вероятность допустить ошибку при его формулировке, а также значительно повышает читаемость запроса для других пользователей. Для примера рассмотрим запрос, возвращающий все листья графа, то есть вершины, не имеющие исходящих ребер, используя обход в глубину. Реализация этого запроса на Gremlin представлена на листинге 15. В начале мы получаем все корни графа, пользуясь итератором по всем вершинам, затем рекурсивно переходим в потомка, избавляясь от повторений, после чего возвращаем встреченные листья.

Листинг 15 – Получение листьев графа на Gremlin

```
g.V().not(in())
  .repeat(out().dedup())
  .until(not(out()))
```

Реализация этого же запроса в нативном виде представлена в приложении В.1 и требует значительно больше строк кода (3 против 32).

Преимущество использования Gremlin также заметно при работе с ребрами, так как WebGraph не предоставляет абстракции над ними, вследствие чего в нативных запросах приходится работать с парами вершин. В качестве примера можно рассмотреть запрос, возвращающий все ребра из вершин типа «снимок» в вершины типа «ревизия», при этом обе вершины должны иметь свойство `timestamp` больше определенного значения. Реализация запроса на Gremlin представлена на листинге 16, а в нативном виде в приложении В.2. Последний требует больше строк кода (4 против 20), а также значительно уступает в читаемости запросу на Gremlin.

Листинг 16 – Получение ребер по типам вершин на Gremlin

```
g.E()
  .and(inV().hasLabel("SNP"),
        outV().hasLabel("REV"),
        both().has("timestamp", lte(123)))
```

Помимо этого при использовании Gremlin упрощается реализация небольших локальных запросов, так как для нативного подхода зачастую необходим повторяющийся шаблонный код. Примером может служить получение значения свойства конкретного ребра. Реализация запроса на Gremlin представлена на листинге 17, а его аналог в нативном виде представлен в приложении В.3.

Листинг 17 – Получение свойства ребра на Gremlin

```
g.E(LongLongImmutablePair.of(fromId, toId)).values()
```

Еще одним преимуществом использования Gremlin является поддержка запуска строковых запросов. Это позволяет, к примеру, создать веб-сервер, принимающий и исполняющий строковые запросы. С учетом удобства написания коротких запросов, такой веб-сервер позволяет очень быстро извлекать произвольную информацию из графа.

Выводы по главе 4

В этой главе было проведено сравнение разработанной библиотеки и нативного подхода к реализации запросов. Для определения потери в эффективности был выбран набор данных, сформулированы запросы, и проведен анализ эффективности их исполнения, показывающий, что получая возможность исполнять Gremlin запросы в своей системе, пользователь столкнется с 4–5 кратным замедлением их исполнения. Также были описаны преимущества, которые получает пользователь при использовании библиотеки.

ЗАКЛЮЧЕНИЕ

В данной работе была описана разработанная библиотека, связывающая фреймворки WebGraph и TinkerPop, и позволяющая осуществлять немодифицирующие Gremlin запросы к любым графам, сжатым фреймворком WebGraph. Это позволяет использовать высокоуровневый доменно-ориентированный язык для формулировки запросов к графу, при этом сохраняя возможность пользоваться эффективностью WebGraph.

Были реализованы интерфейсы Apache TinkerPop, что позволило наладить взаимодействие между Gremlin и WebGraph. Также была добавлена поддержка свойств вершин и ребер, позволяющая формулировать широкий набор запросов. Реализация стандартных подходов к работе со свойствами в WebGraph, реализация исполнителя как строковых Gremlin запросов, так и сформулированных с использованием Gremlin-Java DSL, а также распространение проекта в виде Maven-зависимости позволяет с минимальной настройкой начать пользоваться библиотекой в своей системе.

Эта возможность была использована командой Software Heritage. В рамках работы была реализована поддержка графа репозитория Software Heritage, и добавлена в репозиторий проекта.

Согласно проведенному тестированию было выявлено, что эффективность исполнения Gremlin запросов по сравнению с нативной реализацией снижается в 4–5 раз. При этом разработанная библиотека позволяет формулировать и исполнять более компактные и читаемые, по сравнению с нативными, Gremlin запросы, в том числе в строковом виде, а также впервые позволяет запускать Gremlin запросы на крупных графах на относительно недорогом компьютере.

Дальнейшее развитие проекта возможно в первую очередь в области оптимизаций связки фреймворков. Реализация ряда эвристик может снизить потерю в эффективности, тем самым сделав библиотеку еще более применимой на практике.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 The future is big graphs / S. Sakr [et al.] // Communications of the ACM. — 2021. — Sept. — Vol. 64, no. 9. — P. 62–71. — DOI: 10.1145/3434642. — URL: <https://doi.org/10.1145%5C%2F3434642>.
- 2 *Rodriguez M. A.* The Gremlin Graph Traversal Machine and Language (Invited Talk) // Proceedings of the 15th Symposium on Database Programming Languages. — Pittsburgh, PA, USA : Association for Computing Machinery, 2015. — P. 1–10. — (DBPL 2015). — ISBN 9781450339025. — DOI: 10.1145/2815072.2815073. — URL: <https://doi.org/10.1145/2815072.2815073>.
- 3 *Boldi P., Vigna S.* The WebGraph Framework I: Compression Techniques // Proc. of the Thirteenth International World Wide Web Conference (WWW 2004). — Manhattan, USA : ACM Press, 2004. — P. 595–601.
- 4 *Neo4j.* Graph Database Scalability [Электронный ресурс]. — 2022. — URL: <https://neo4j.com/product/neo4j-graph-database/scalability/>.
- 5 *McSherry F., Isard M., Murray D. G.* Scalability! But at What Cost? // Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems. — Switzerland : USENIX Association, 2015. — P. 14. — (HOTOS'15).
- 6 *Pietri A.* Organizing the graph of public software development for large-scale mining : Theses / Pietri Antoine. — Inria, 11/2021. — URL: <https://hal.archives-ouvertes.fr/tel-03515795>.
- 7 Ultra-Large-Scale Repository Analysis via Graph Compression / P. Boldi [et al.] // SANER 2020: The 27th IEEE International Conference on Software Analysis, Evolution and Reengineering. — IEEE, 2020. — P. 184–194. — DOI: 10.1109/SANER48275.2020.9054827.
- 8 Software Heritage [Электронный ресурс]. — 2022. — URL: <https://www.softwareheritage.org>.
- 9 TinkerPop Gremlin backend for WebGraph (internship) [Электронный ресурс]. — 2022. — URL: [https://wiki.softwareheritage.org/wiki/TinkerPop_Gremlin_backend_for_WebGraph_\(internship\)](https://wiki.softwareheritage.org/wiki/TinkerPop_Gremlin_backend_for_WebGraph_(internship)).

- 10 Data System Providers [Электронный ресурс]. — 2022. — URL: <https://tinkerpop.apache.org/providers.html>.
- 11 Apache TinkerPop [Электронный ресурс]. — 2022. — URL: <https://tinkerpop.apache.org>.
- 12 Git [Электронный ресурс]. — 2022. — URL: <https://git-scm.com>.
- 13 Dataset - Software Heritage [Электронный ресурс]. — 2022. — URL: <https://docs.softwareheritage.org/devel/swh-dataset/graph/dataset.html>.
- 14 Graph RPC API - Software Heritage [Электронный ресурс]. — 2022. — URL: <https://docs.softwareheritage.org/devel/swh-graph/api.html>.
- 15 Add BidirectionalImmutableGraph to hold a graph and its transpose [Электронный ресурс]. — 2022. — URL: <https://github.com/vigna/webgraph-big/commit/debeb714e1392c5d171fea784115de63c1086170>.
- 16 fastutil [Электронный ресурс]. — 2022. — URL: <https://fastutil.di.unimi.it>.
- 17 Apache Maven [Электронный ресурс]. — 2022. — URL: <https://maven.apache.org>.
- 18 swh-graph [Электронный ресурс]. — 2022. — URL: <https://forge.softwareheritage.org/source/swh-graph>.
- 19 TinkerPop Documentation [Электронный ресурс]. — 2022. — URL: <https://tinkerpop.apache.org/docs/current/reference>.
- 20 Laboratory of Web Algorithmics [Электронный ресурс]. — 2022. — URL: <https://law.di.unimi.it/datasets.php>.

ПРИЛОЖЕНИЕ А. ИСПОЛНИТЕЛЬ GREMLIN ЗАПРОСОВ

Листинг А.1 – Реализация метода, исполняющего строковые Gremlin запросы

```
public GraphTraversal<?, ?> eval(String query) {
    Bindings bindings = new SimpleBindings();
    bindings.put("g", g.traversal());
    try (GremlinExecutor ge =
        GremlinExecutor.build()
            .globalBindings(bindings)
            .create()) {
        return (GraphTraversal<?, ?>) ge.eval(query, bindings).get();
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

ПРИЛОЖЕНИЕ Б. ПРИМЕР ИСПОЛЬЗОВАНИЯ БИБЛИОТЕКИ ДЛЯ ДОБАВЛЕНИЯ СВОЙСТВ

Листинг Б.1 – Добавление свойств для графа Software Heritage

```
StandardWebGraphPropertyProvider provider = new
    StandardWebGraphPropertyProvider();
provider.setVertexLabeller(id -> graph.getNodeType(id).toString())
;
provider.addVertexProperty(new FileVertexProperty<>("
    author_timestamp", Long.class, Path.of(path + ".property.
    author_timestamp.bin")));
provider.addVertexProperty(new VertexProperty<>("swhid", graph::
    getSWHID));
provider.addVertexProperty(new VertexProperty<>("message", nodeId
    -> {
        try {
            return graph.getProperties().getMessage(nodeId);
        } catch (IOException e) {
            throw new UncheckedIOException(e);
        }
    }));
ArcLabelEdgeProperty<DirEntry[]> edgeProperty = new
    ArcLabelEdgeProperty<>(graph.getForwardGraph().
    underlyingLabelledGraph());
provider.addEdgeProperty(edgeProperty);
provider.addEdgeProperty(new ArcLabelEdgeSubProperty<>("
    dir_entry_str", edgeProperty, SwhProperties::dirEntryStr));
```

ПРИЛОЖЕНИЕ В. ПРИМЕРЫ НАТИВНЫХ ЗАПРОСОВ

Листинг В.1 – Получение листьев графа нативным способом

```
public Set<Long> leaves(SwhBidirectionalGraph g) {
    NodeIterator nodes = g.nodeIterator();
    Set<Long> roots = new HashSet<>();
    while (nodes.hasNext()) {
        long cur = nodes.nextLong();
        if (g.predecessors(cur).nextLong() == -1) {
            roots.add(cur);
        }
    }
    Set<Long> leaves = new HashSet<>();
    boolean[][] used = BooleanBigArrays.newBigArray(g.numNodes());
    for (Long root : roots) {
        dfs(root, used, leaves, g);
    }
    return leaves;
}

private void dfs(long root, boolean[][] used, Set<Long> leaves,
    BidirectionalImmutableGraph g) {
    BigArrays.set(used, root, true);
    LazyLongIterator successors = g.successors(root);
    long child;
    boolean hasChild = false;
    while ((child = successors.nextLong()) != -1) {
        hasChild = true;
        if (!BigArrays.get(used, child)) {
            dfs(child, used, leaves, g);
        }
    }
    if (!hasChild) {
        leaves.add(root);
    }
}
```

Листинг В.2 – Получение ребер по типам вершин нативным способом

```

public Set<LongLongPair> snpRelEdges(SwhBidirectionalGraph g) {
    NodeIterator nodes = g.nodeIterator();
    Set<LongLongPair> edges = new HashSet<>();
    while (nodes.hasNext()) {
        long from = nodes.nextLong();
        if (g.getNodeType(from) != Node.Type.SNP || g.getTimestamp
            (from) > 123) {
            continue;
        }
        LazyLongIterator successors = g.successors(from);
        long to;
        while ((to = successors.nextLong()) != -1) {
            if (g.getNodeType(to) == Node.Type.REV && g.
                getAuthorTimestamp(to) <= 123) {
                edges.add(LongLongPair.of(from, to));
            }
        }
    }
    return edges;
}

```

Листинг В.3 – Получение свойства ребра нативным способом

```

public Object edgeProperty(long fromId, long toId,
    SwhBidirectionalGraph g) {
    ArcLabelledNodeIterator.LabelledArcIterator s = g.
        labelledSuccessors(fromId);
    long succ;
    while ((succ = s.nextLong()) != -1) {
        if (succ == toId) {
            return s.label().get();
        }
    }
    return null;
}

```