Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Визуализация данных блокчейна Биткоин

Студент ИУ9-51Б _____ _А.С. Яровикова___
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ _И.Э. Вишняков_____
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2022 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Блокчейн — технология, которая позволяет обеспечить защищенность и достоверность данных, сохраненных в формате упорядоченной цепочки блоков. На основе данной технологии реализована известная пиринговая платежная система Биткоин [1-2]. Одноименная единица счета, используемая для учета операций перевода в сети, является первой и лидирующей криптовалютой в мире. Вследствие чего, сеть имеет огромное количество информации, которую сложно анализировать в ее исходном виде. Для упрощения анализа данных транзакций сети Биткоин удобно иметь представление в интуитивно понятной визуальной форме. Универсальным средством для такого представления являются графы [3-4].

Целью данной курсовой работы является разработка программы для визуализации данных блокчейна Биткоин на основе графовой модели.

1. Обзор предметной области

1.1. Методы визуализации графов

Особенностью задачи визуализации графов является направленность на пользовательские потребности. Пользователь сам определяет свойства моделируемых объектов, которые будут отображаться в графическом представлении, и требования к ним.

В зависимости от требований к модели один и тот же граф можно визуализировать разными способами. Для формализации понятия качественного способа рисования графа принято руководствоваться изобразительными соглашениями о размещении вершин и ребер графа, а также эстетическими критериями [5].

Широко применяются следующие размещения:

- полинейное – ребра в виде ломаных линий;
- произвольное;
- прямолинейное – ребра в виде отрезков прямой;
- ортогональное – ребра из ломаных линий, состоящих из чередующихся горизонтальных и вертикальных сегментов;
- сетчатое – все вершины, сгибы и точки пересечения ребер имеют целочисленные координаты;
- плоское (планарное) – у ребер отсутствуют точки пересечения;
- восходящее (или нисходящее) – для орграфов, где координаты точек ребер монотонно изменяются снизу вверх (или сверху вниз) и слева направо.

Примерами эстетических критериев являются:

- минимизация пересечений и сгибов ребер;
- минимизация площади размещения;
- минимизация (иногда унификация) длины ребер;
- максимизация симметричности изображения.

Следует понимать, что большинство эстетических критериев являются сложными задачами оптимизации с вычислительной точки зрения. Поэтому

часто удовлетворить одновременно всем критериям является невозможным. Вследствие чего, существует множество различных методов визуализации графов. Более того, задача выбора метода визуализации полностью основана на особенностях конкретной выполняемой задачи.

Наиболее широко распространенными способами изображения графов являются методы визуализации неориентированных графов на основе физических аналогий, круговые алгоритмы, иерархические и поуровневые методы, ортогональные методы [6]. Описанные типы изображений показаны на рисунке 1.

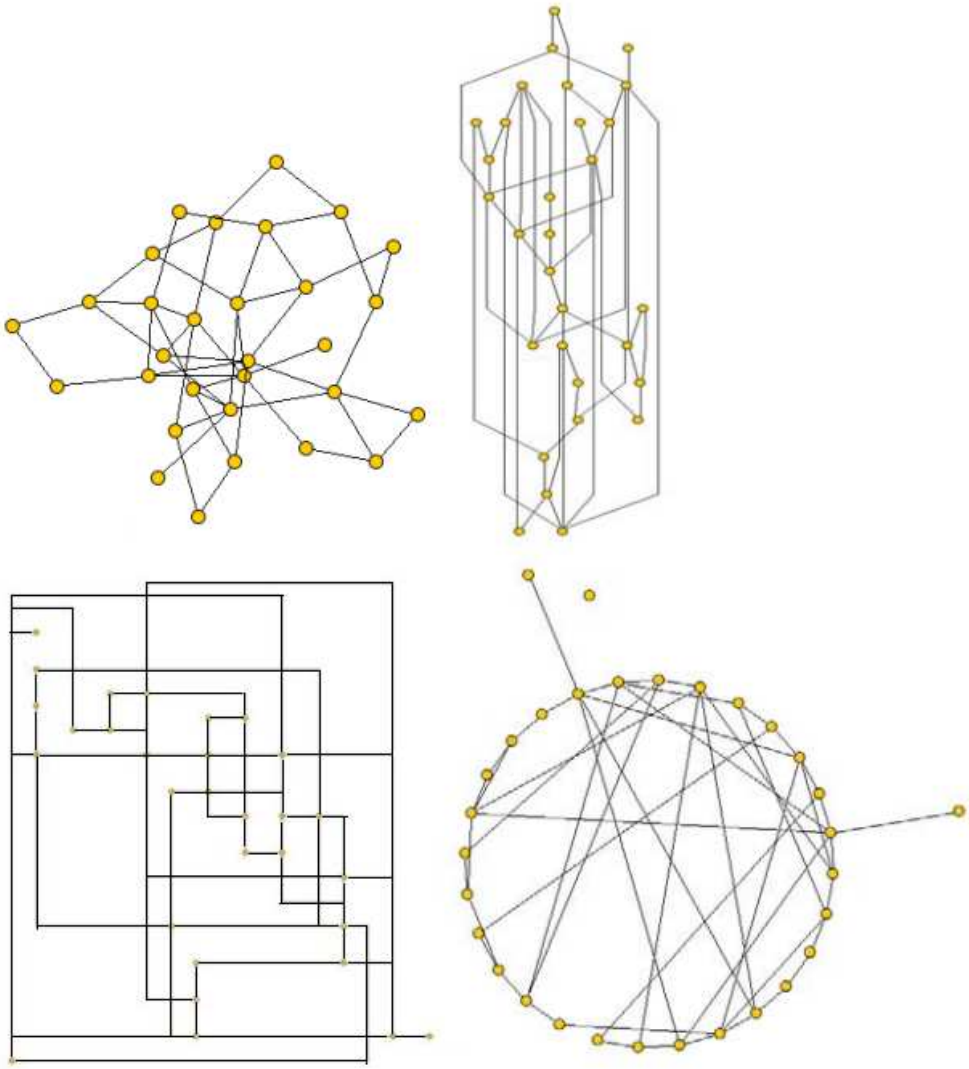

Рисунок 1. Общепринятые статические изображения графов

1.2. Применение графов при анализе Биткоин

Графовые модели позволяют представить любую информацию, которую можно промоделировать в виде объектов и связей между ними. Поэтому графы широко используются в различных областях науки, в том числе для визуализации технологии блокчейн.

Данные сети Биткоин хранятся в виде цепочки блоков, внутри которых содержится информация о проведенных транзакциях. Именно поэтому удобно представлять большой объем данных блокчейна Биткоин в виде графовой модели, распределив блоки, транзакции и связи между ними в разные объекты, связанные между собой.

В качестве хранилища для данных сети Биткоин изначально решено использовать базу данных, созданную в мультимодельной СУБД ArangoDB. Данная система дает возможность распределить большие данные о вершинах и ребрах графа в отдельные коллекции и работать с ними с помощью SQL-подобного языка запросов AQL [7].

2. Разработка приложения визуализации данных Биткоин

2.1. Требования к представлению данных Биткоин

Основным требованием для графа является его древовидное размещение слева направо, поскольку для визуализации данных блокчейна Биткоин большое значение имеет хронология появления транзакций. Таким образом, слева должны располагаться более ранние транзакции, а справа транзакции, совершенные позднее.

Полученные данные о транзакциях Биткоин должны храниться в базе данных в системе ArangoDB. Её схема представлена на рисунке 2. Вершинам графа должны соответствовать таблицы Block, Tx, Address, ребрам – таблицы parentBlock, next, in, out. При этом таблица in соответствует ребру, входящему в вершину Address, а таблица out соответствует ребру, выходящему из вершины Address. Для хранения всех вершин и ребер созданы коллекции в ArangoDB соответственно btcBlock, btctx, btcAddress и btcParentBlock, btcNext, btcIn, btcOut.

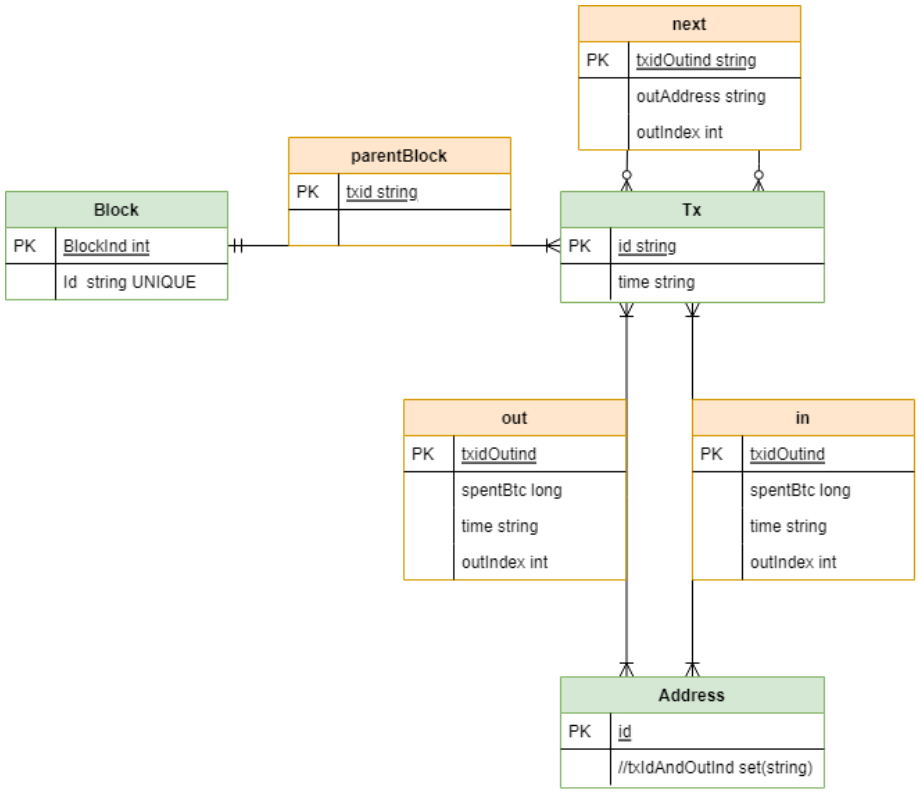

Рисунок 2. Схема представления графа Биткоин в ArangoDB

Важным требованием для визуализации данных является возможность различать вершины и ребра разных коллекций. Покраска вершин и ребер в различные цвета в соответствии с принадлежностью к коллекции является решением этой задачи.

Блокчейн Биткоин имеет огромное количество данных, анализ которых проводится последовательно, путем сбора информации о связанных блоках и транзакциях. Поэтому создавать сразу визуализацию всего графа, начиная с первой транзакции первого блока, не имеет смысла. Вследствие чего возникает следующий ряд требований к реализации:

- построение графа, начиная с заданной пользователем транзакции;
- подпись вершин графа соответствующими идентификаторами;
- возможность перемещения вершин графа;
- возможность просмотра информации о любой вершине, любом ребре;
- возможность свернуть вершину по клику (т.е. вершины, связанные с данной и находящиеся правее, не должны отображаться в графе);
- возможность раскрыть вершину (действие, обратное сворачиванию).

2.2. Формат разрабатываемого приложения

В целях удобства работы пользователя было решено разработать приложение, состоящее из двух модулей. Один модуль представляет собой пользовательский интерфейс, через который происходит построение графа и интерактивное взаимодействие с ним на основе введенных начальных данных. В соответствии с поставленными требованиями в интерфейсе должны быть предусмотрены:

- поле для ввода пользователем корня будущего графа;
- поля для задания цвета для каждого типа вершин и ребер;
- поле для просмотра детальной информации о выбранной вершине или выбранном ребре графа.

Второй модуль приложения является модулем обработки запросов, полученных от пользовательского интерфейса. Его основная задача

заключается в получении значения корня будущего графа и подключении к локальной базе данных в ArangoDB для поиска связанных с ним вершин. Таким образом, модуль обработки запросов должен реализовывать построение структуры будущего графа, после чего отправлять данные в первый модуль, где и будет происходить визуализация графа.

При таком подходе в случае изменения схемы базы данных ArangoDB в приложении необходимо только изменить запрос к базе данных и перезапустить модуль пользовательского интерфейса для задания нового корня графа.

2.3. Особенности визуализации графа данных Биткоин

Согласно основным требованиям к представлению графа, он должен иметь древовидное размещение слева направо и корнем графа должна являться заданная пользователем транзакция. Более того, важным критерием качественного представления графа транзакций является соответствие топологии сети Биткоин: отслеживаемые средства адреса одного кошелька посредством транзакций распределяются по другим кошелькам, из которых в свою очередь средства могут быть отправлены на адреса других кошельков пользователей сети. Эта цепочка событий продолжается до тех пор, пока вся начальная сумма не распределится по другим счетам. Заметим, что зачастую средства, отправленные с одного адреса, не возвращаются обратно в полном объеме, поскольку распределяются между другими адресами пользователей сети Биткоин. Упомянутые требования позволяют сохранить описанную топологию. Таким образом, граф является ориентированным, а для наглядной визуализации связанных транзакций выбран нисходящий метод представления. Для сохранения эстетичности визуализации выбран дополнительный (а потому не всегда выполнимый) критерий унификации длины ребер между вершинами транзакций графа. На рисунке 3 показан граф, обладающий перечисленными свойствами.

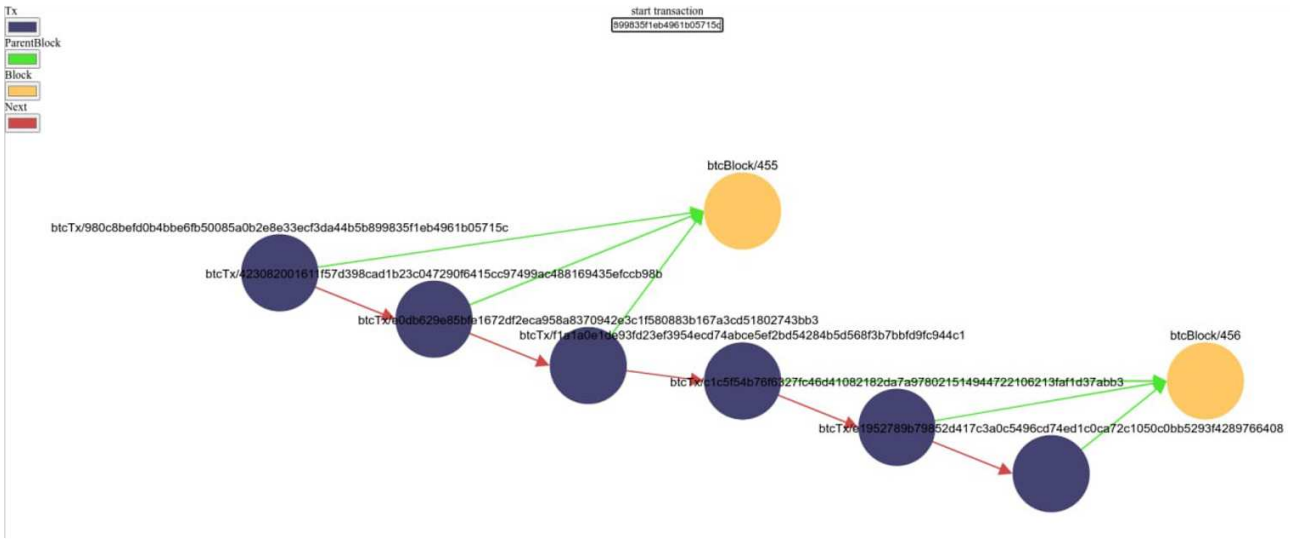

Рисунок 3. Граф блокчейна Биткоин с нисходящим расположением транзакций и равными ребрами между ними

Другой важной особенность визуализации графа является возможность наличия «свернутых» вершин. Сворачивание вершин необходимо разработать для удобства использования приложения и анализа графа. Это действие над заданной вершиной подразумевает визуальное удаление всех вершин, связанных с заданной и находящихся правее. Для наглядности решено отмечать «свернутые» вершины знаком плюса «+». Так, например, на рисунке 4 проведено сворачивание вершины 3. Операция, обратная к сворачиванию является операцией разворачивания. Обе операции обрабатываются согласно алгоритму, описанному ниже.

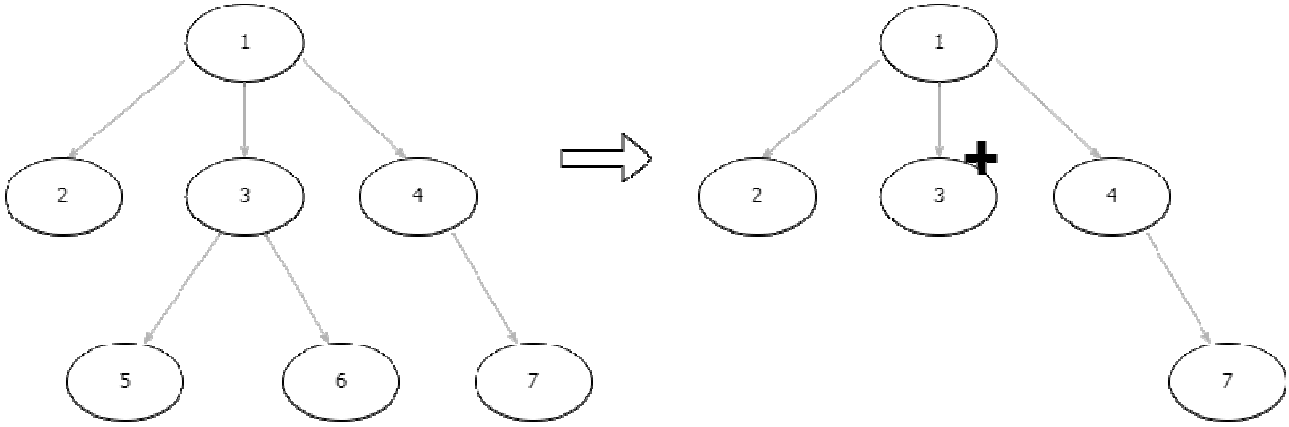

Рисунок 4. Иллюстрация операции сворачивания вершины

Операция, обратная к сворачиванию является операцией разворачивания. Обе операции обрабатываются согласно алгоритму:

1. Сохранить состояние текущей вершины. Состояние определяет, свернута вершина, или нет;
2. Проверить состояние вершины:
   2.1. Если вершина не свернута, то свернуть вершину и сменить значение состояния на противоположное значение;
   2.2. Если вершина свернута, то развернуть вершину и сменить значение состояния на противоположное значение.

2.4. Особенности структуры хранения графа данных Биткоин

Получение и хранение данных Биткоин обеспечивает модуль обработки запросов. Получение данных происходит путем считывания графа из базы данных, начиная с корневой (стартовой) вершины, и последовательным сохранением их в какой-то объект, инициализирующий граф. Соответственно предполагается наличие этой вершины в базе данных. А также для работы алгоритма предполагается, что существуют объекты коллекций, соответствующие всем типам вершин и ребер графа.

Алгоритм считывания графа:

1. Добавить стартовую вершину в объект графа;
2. По всем коллекциям ребер:
   2.1. Если текущая коллекция не btcNext и стартовая вершина имеет ребро из текущей коллекции:
      2.1.1. Добавить в объект графа это ребро и вершину, с которой ребро связывает стартовую вершину.
   2.2. Если текущая коллекция является коллекцией ребер между транзакциями (btcNext) и стартовая вершина имеет ребро из текущей коллекции:

2.2.1. Добавить в объект графа это ребро;

2.2.2. Для вершины, с которой связывает это ребро, пройти по всем коллекциям ребер для поиска связей (пункт 2 алгоритма).

В процессе считывания графа происходит сохранение его вершин и ребер в некоторый объект, инициализирующий граф. Структура этого объекта должна быть подобрана таким образом, чтобы работать с ней было удобно и эффективно.

Основные требования к структуре данных для хранения графа:

- Определение типа вершины и типа ребра за O(1);
- Получение подробной информации о вершине или ребре за O(1);
- Сохранение порядка очередности вершин и ребер, в которой они считывались с базы;
- Переход к дочерней или родительской вершине за O(1).

Таким образом, было решено использовать для каждой коллекции вершин и ребер структуры (или классы, если язык реализации поддерживает объектно-ориентированное программирование), внутри которых определены поля соответствующие свойствам вершин и классов. Это позволяет получить тип вершины, тип ребра, дочернюю и родительскую вершину, связанное с вершиной ребро за константное время. Для сохранения очередности вершин при визуализации графа решено хранить вершины и ребра в списке объектов для визуализации.

3. Реализация приложения визуализации данных Биткоин

Для выполнения поставленной задачи было необходимо реализовать два модуля приложения: пользовательский интерфейс и обработчик пользовательских запросов на построение графа.

Модуль пользовательского интерфейса получает значение корня графа и передает модулю обработки, реализует корректное размещение вершин и ребер графа при визуализации, операции сворачивания и разворачивания вершин, создание подписей около вершин, а также покраску вершин и ребер в соответствии с их типом. Таким образом, основная задача первого модуля заключается в реализации интерактивного взаимодействия пользователя с графом.

Модуль обработки запросов отвечает за получение данных о корне графа, корректное считывание графа из базы данных и структурированное хранение вершин и ребер графа для дальнейшей передачи их первому модулю для визуализации.

Разработка и сборка проекта осуществлялась в среде разработки Intellij IDEA [8]. Эта среда разработки предоставляет умное автодополнение, анализ кода и надежный рефакторинг – возможности, необходимые для быстрой и эффективной разработки.

3.1. Особенности реализации модуля обработки запросов

Было решено начать с данного модуля, поскольку необходимо реализовать получение данных Биткоин из внутреннего хранилища. Модуль обработки запросов на построение графа должен получать данные из базы в системе ArangoDB, поэтому в выборе языка реализации решено отталкиваться от существующих драйверов для данной СУБД.

Идеальным решением оказался язык Java [9] благодаря тому, что язык объектно-ориентированный, а также наличию официального драйвера для ArangoDB и интеграции с фреймворком Spring [10]. Данный фреймворк используется разработчиками для облегчения проектирования и создания

приложений на языке Java. Набор инструментов Spring для построения зависимостей позволяет сосредоточиться на логике проекта, а дополнение фреймворка Spring Boot [11] автоматически конфигурирует проект и настраивает его компоненты с помощью аннотаций. Однако для использования интеграции Spring Data ArangoDB необходимо подключать эту зависимость при сборке проекта. Для автоматизации процесса сборки используется фреймворк Apache Maven [12], осуществляющий вызов компилятора и автоматическое управление зависимостями и ресурсами проекта.

3.2. Особенности реализации модуля пользовательского интерфейса

В качестве библиотеки для визуализации графа была выбрана Cytoscape.js [13], используемая для визуализации сложных сетей и интеграции их с любыми типами данных атрибутов. Данная библиотека имеет несколько возможных реализаций макетов графов. Для сохранения древовидного представления графа Биткоин был выбран макет darge [14]. Выбранный макет также дает возможность реализовать подписи к вершинам, перемещение пользователем вершин графа и модифицировать направления ребер. Таким образом, получилось создать нисходящее расположение вершин транзакций графа и частичное интерактивное взаимодействие пользователя с ним.

Для создания удобного пользовательского интерфейса была создана веб-страница с помощью языка разметки HTML и стилей CSS. С помощью языка JavaScript полностью реализованы пользовательские возможности по просмотру детальной информации о вершине или ребре, сворачиванию и разворачиванию вершин и покраске вершин и ребер в различные цвета в соответствии с их типом.

Для реализации передачи полученных от пользователя данных на второй модуль и обратной передачи элементов графа для визуализации были использованы технология Node.js [15], позволяющая писать серверный код для веб-страниц, и HTTP-клиент Axios.js [16], основанный на объектах, содержащих результат асинхронных операций, для Node.js. Таким образом, с

помощью Axios реализовано получение списка элементов графа для визуализации, отправленных с модуля обработки запросов на построение графа.

3.3. Описание работы приложения

Разработанное приложение работает следующим образом:

1. Пользователь заполняет конфигурационный файл (см. рисунок 5) для корректного связывания модуля обработки запросов с базой в ArangoDB
2. Модуль обработки запросов связывается с базой данных и получает данные каждой коллекции.
3. Пользовательский интерфейс ожидает ввода корневой вершины будущего графа. После получения данных модуль передает их на сторону модуля обработки запросов.
4. Модуль обработки запросов получает данные и согласно алгоритму считывания графа производит чтение графа, начиная с полученной корневой вершины, и формирует список элементов графа для визуализации (в листинге 1 приложения А приведен код реализации считывания графа).
5. Пользовательский интерфейс запрашивает у модуля обработки запросов данные для визуализации графа. Когда данные получены, модуль пользовательского интерфейса создает визуальное представление графа (см. рисунок 6).
6. Пользовательский интерфейс обрабатывает события пользователя:
   6.1. Нажатие на вершину/ребро – на время сохранения нажатия вершина/ребро выделяется путем появления серого прямоугольника, в правом верхнем углу интерфейса выводится детальная информация о вершине/ребре в формате JSON (см. рисунок 7).
   6.2. Двойное нажатие на вершину – производится сворачивание или разворачивание вершины в зависимости от текущего состояния вершины. Если вершина свернута, она отмечена знаком плюс «+» (см.

рисунок 8). Код реализации данных операций приведен в листинге 2 приложения А.

6.3. Прокрутка колеса мыши – масштабирование графа (см. рисунок 9)

```
application.properties M
src > main > resources > application.properties
1  arangodb.spring.data.database=mybd
2  arangodb.spring.data.user=root
3  arangodb.spring.data.password=
4
```

Рисунок 5. Пример конфигурационного файла для связывания с базой данных

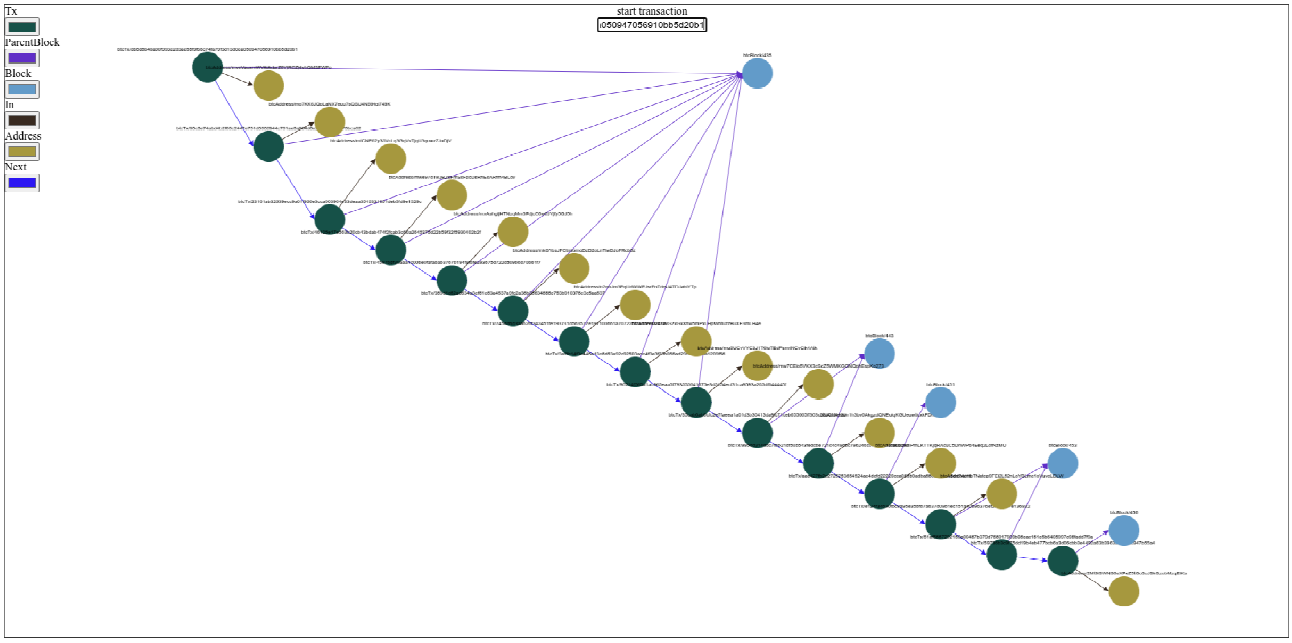

Рисунок 6. Первичное размещение графа

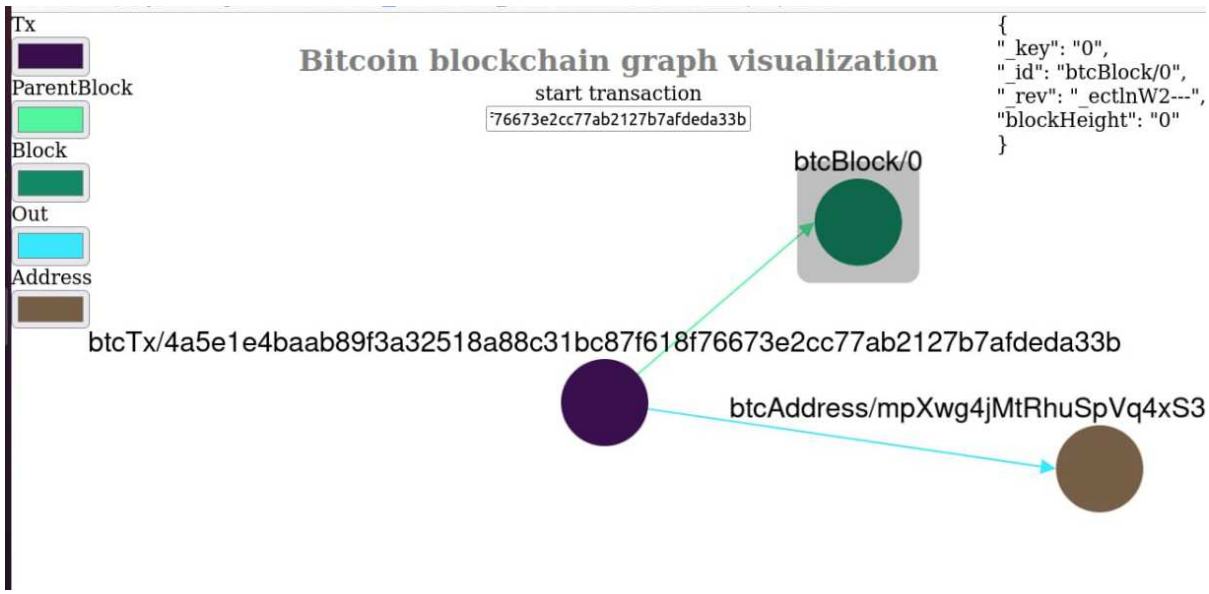

Рисунок 7. Операция нажатия на вершину

Рисунок 8. Операция сворачивания вершины

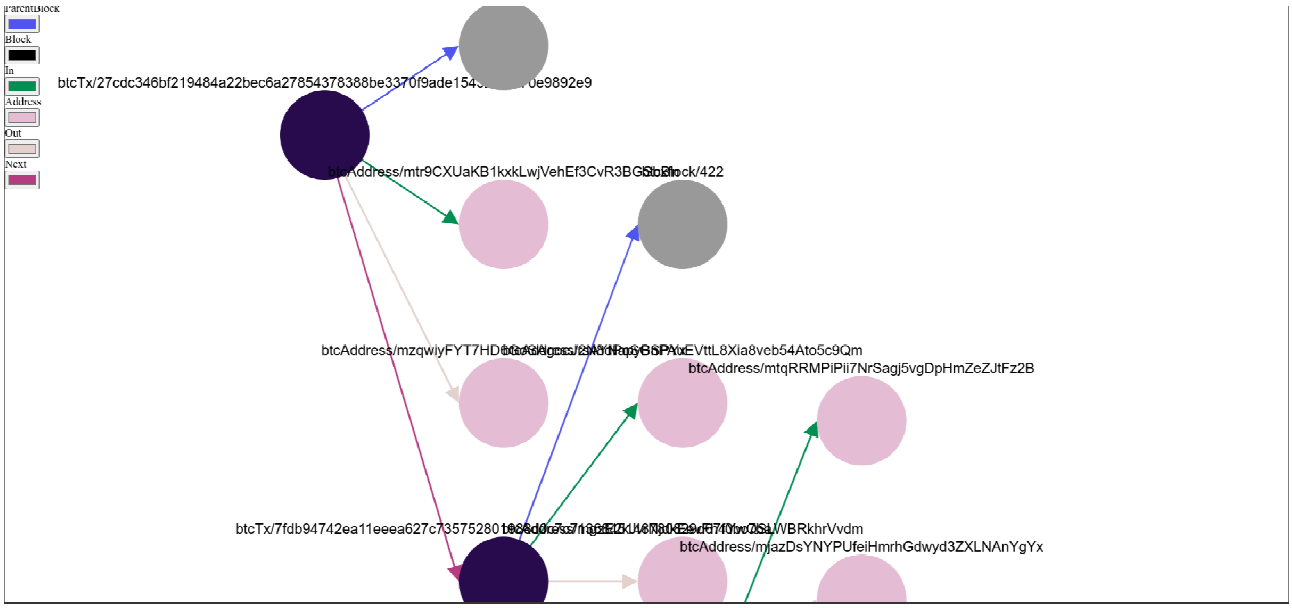

Рисунок 9. Прокрутка колеса мыши

4. Тестирование

Цель тестирования заключается в оценке производительности разработанной программы и поиске недочетов реализации, выражающихся в неудобстве пользования.

Изначально тестирование проводилось на реальных данных тестовой версии блокчейна Биткоин. Но поиск транзакций с определенным количеством связанных с ней вершин стал затруднительным, с увеличением глубины графа. Поэтому для создания больших по глубине графов была создана отдельная программа на языке Go [16], генерирующая случайный граф, с заданным количеством вершин, соответствующих коллекциям базы данных ArangoDB.

Производительность программы оценивается по результатам замеров времени на этапах ответа модуля обработки запросов и времени визуализации графа.

В связи с особенностями динамической JIT-компиляции [17], применяемой в Java-приложениях, для достижения точности измерений каждый тест запускался 3 раза, а округленное среднее значение заносилось в отчет. Результаты тестирования приведены в таблице 1.

Тестирование производилось на машине с характеристиками:

- ОС: Ubuntu 20.04.5
- ЦП: Intel(R) Core(TM) i5-8250U
- Тактовая частота: 1.60GHz

Таблица 1 – Результаты тестирования производительности

Количество вершин	Глубина графа	Время ответа модуля обработки запросов, с.	Время визуализации графа, с.
3	2	0,04	0,01
7	5	0,21	0,03
34	15	0,41	0,06
40	15	0.43	0,07
120	110	0,71	0,42
300	275	2,14	1,34
600	550	5,01	3,82
1200	1100	17,16	14,96
2400	2200	21,34	15,01

По данным, полученным в результате тестирования, можно сделать вывод о том, что с увеличением глубины графа (от 200 и более) значительно растет время обработки запроса на построение графа и время самой визуализация. Прежде всего, это обусловлено особенностью считывания данных из базы в ArangoDB и структуры хранения данных графа. Стоит отметить, что на небольших по глубине графах процесс визуализации занимает в среднем в разы меньше времени, чем заполнение структуры графа в модуле обработки запросов. Повторим, что данное приложение разрабатывается в первую очередь для работы с небольшими подграфами сети Биткоин, а потому данный недостаток не является критичным. Основную цель программа выполняет исправно.

Удобство использования является достаточно субъективным критерием, а потому его тестирование трудно оценивать. Однако путем симуляции работы с приложением были выявлены некоторые недостатки пользовательского интерфейса, которые были учтены при написании данной работы.

ЗАКЛЮЧЕНИЕ

В результате курсовой работы поставленная задачи была выполнена, и было разработано приложение для визуализации данных блокчейна Биткоин на основе графовой модели. В процессе выполнения работы были исследованы различные методы визуализации графов и технология блокчейн. Удалось разработать и реализовать алгоритмы для получения информации из базы данных и создания по ней визуализации графа. В ходе разработки был получен опыт работы с СУБД ArangoDB, фреймворками Spring и Apache Maven, а также платформой Node.js, HTTP-клиентом Axios.js и библиотекой Cytoscape.js.

В дальнейшем необходимо совершенствование приложения и оптимизация алгоритмов, поскольку тестирование показало недостатки в виде низкой скорости работы на графах с количеством вершин более тысячи. Также приложение можно доработать, добавив возможности построения графа с произвольной вершины любого типа и возможности топологической сортировки графа по заданному пользователем признаку.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Официальный сайт Биткоин – URL: bitcoin.org/ru/how-it-works (дата обращения: 19.01.2023).
2. Nakamoto S., Bitcoin: A Peer-to-Peer Electronic Cash System. – 2008.
3. Касьянов В. Н. Применение графов в программировании // Программирование. – 2001. - N 3. - С. 51-70.
4. Касьянов В. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003.
5. Касьянов В. Н., Касьянова Е. В. Визуализация графов и графовых моделей. - Новосибирск: Сибирское Научное Издательство, 2010.
6. Апанович З. В. От рисования графов к визуализации информации. – Новосибирск, 2007. – 27 с. (Препр. / ИСИ СО РАН; № 148).
7. ArangoDB – URL: arangodb.com (дата обращения: 19.01.2023).
8. Intellij IDEA: IDE для профессиональной разработки на Java от JetBrains – URL: jetbrains.com/ru-ru/idea (дата обращения 19.01.2023).
9. Что такое Java? – URL: java.com/ru/about/whatis_java.jsp (дата обращения 20.01.2023).
10. Spring Framework – URL: spring.io/projects/spring-framework (дата обращения 20.01.2023).
11. Spring Boot – URL: spring.io/projects/spring-boot (дата обращения 20.01.2023).
12. Apache Maven – URL: maven.apache.org (дата обращения 20.01.2023).
13. What is Cytoscape? – URL: cytoscape.org/what_is_cytoscape.html (дата обращения 20.01.2023).
14. The Darge layout for Cytoscape.js – URL: github.com/cytoscape/cytoscape.js-dagre (дата обращения 20.01.2023).
15. Все про Node.js – URL: ru.hexlet.io/blog/posts/zachem-izuchat-node-js-ili-o-perspektivah-bekenda-na-javascript (дата обращения 20.01.2023).

16. Что такое Axios? – URL: axios-http.com/ru/docs/intro (дата обращения 20.01.2023).
17. Официальный сайт Go – URL: go.dev (дата обращения: 20.01.2023).
18. How the JIT compiler optimizes code – URL: ibm.com/docs/en/sdk-java-technology/8?topic=compiler-how-jit-optimizes-code (дата обращения: 20.01.2023).

ПРИЛОЖЕНИЕ А

Листинг 1 – Считывание графа из объектов коллекций ArangoDB

```
private List<CytoscapeDataElementDto>
getElementsRelatedWithTx(Tx startTx) {
    List<CytoscapeDataElementDto>
cytoscapeDataElementDtos = new ArrayList<>();

cytoscapeDataElementDtos.add(CytoscapeNodeDto.builder()
                .id(startTx.get_id())
                .type(startTx.getClass().getSimpleName())
                .description(startTx)
                .build());
        try {
            Optional<ParentBlock> optionalParentBlock =
parentBlockRepository.findBy_from__id(startTx.get_id());
            if (optionalParentBlock.isPresent()) {
                ParentBlock parentBlock =
optionalParentBlock.get();
                Optional<Block> optionalBlock =
Optional.ofNullable(parentBlock.get_to());
                if (optionalBlock.isPresent()) {
                    Block block = optionalBlock.get();

cytoscapeDataElementDtos.add(CytoscapeEdgeDto.builder()
                            .source(startTx.get_id())
                            .target(block.get_id())

.type(parentBlock.getClass().getSimpleName())
                            .description(parentBlock)
                            .build());

cytoscapeDataElementDtos.add(CytoscapeNodeDto.builder()
                            .id(block.get_id())

.type(block.getClass().getSimpleName())
```

```
                                .description(block)
                                .build());

                    }
                }
            } catch (MappingException ignored) {
            }
            try {
                Optional<In> optionalIn =
inRepository.findBy_to__id(startTx.get_id());
                if (optionalIn.isPresent()) {
                    In in = optionalIn.get();
                    Optional<Address> optionalAddress =
Optional.ofNullable(in.get_from());
                    if (optionalAddress.isPresent()) {
                        Address address =
optionalAddress.get();

cytoscapeDataElementDtos.add(CytoscapeEdgeDto.builder()
                                .source(startTx.get_id())
                                .target(address.get_id())

.type(in.getClass().getSimpleName())
                                .description(in)
                                .build());

cytoscapeDataElementDtos.add(CytoscapeNodeDto.builder()
                                .id(address.get_id())

.type(address.getClass().getSimpleName())
                                .description(address)
                                .build());
                    }
                }
            } catch (MappingException ignored) {
```

```
        }
        try {
            Optional<Out> optionalOut =
outRepository.findBy_from__id(startTx.get_id());
            if (optionalOut.isPresent()) {
                Out out = optionalOut.get();
                Optional<Address> optionalAddress =
Optional.ofNullable(out.get_to());
                if (optionalAddress.isPresent()) {
                    Address address =
optionalAddress.get();

cytoscapeDataElementDtos.add(CytoscapeEdgeDto.builder()
                            .source(startTx.get_id())
                            .target(address.get_id())

.type(out.getClass().getSimpleName())
                            .description(out)
                            .build());

cytoscapeDataElementDtos.add(CytoscapeNodeDto.builder()
                            .id(address.get_id())

.type(address.getClass().getSimpleName())
                            .description(address)
                            .build());
                }
            }
        } catch (MappingException ignored) {
        }
        try {
            Optional<Next> optionalNext =
nextRepository.findBy_from__id(startTx.get_id());
            if (optionalNext.isPresent()) {
                Next next = optionalNext.get();
```

```
                Optional<Tx> optionalTx =
Optional.ofNullable(next.get_to());
                if (optionalTx.isPresent()) {
                    Tx tx = optionalTx.get();

cytoscapeDataElementDtos.add(CytoscapeEdgeDto.builder()
                            .source(startTx.get_id())
                            .target(tx.get_id())

.type(next.getClass().getSimpleName())
                            .description(next)
                            .build());

cytoscapeDataElementDtos.addAll(getElementsRelatedWithTx(
tx));
                }
            }
        } catch (MappingException ignored) {
        }
        return cytoscapeDataElementDtos;
    }
```

Листинг 2 – Операция сворачивания и разворачивания вершины

```
const hideSuccessors = (node) => {
    node.toggleClass("collapsed")
    if (node.hasClass("collapsed")) {
          node.successors().addClass("hidden");
    } else {
          node.successors().removeClass("hidden");
    }
}
cy.on('dblclick', 'node', function (e) {
hideSuccessors(e.target);
});
```