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 «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Визуализация графа связей пользователей социальной сети»

Студент ИУ9-51Б (Группа) _____ (Подпись, дата) Киселев К.А. (И.О. Фамилия)

Руководитель _____ (Подпись, дата) Каганов Ю.Т. (И.О. Фамилия)

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

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В эпоху цифровых коммуникаций социальные сети играют ключевую роль в повседневной жизни, предоставляя своим пользователям возможность активного взаимодействия и обмена информацией. С ростом объема данных, сгенерированных социальными платформами, возникает неотложная потребность в разработке эффективных методов анализа, позволяющих понять сложные взаимосвязи между участниками сети. В данном контексте визуализация связей пользователей становится ключевым инструментом, обеспечивающим наглядное отображение структуры социальных взаимодействий.

Целью данной курсовой работы является разработка программы для визуализации данных о связях пользователей социальной сети на основе графовой модели.

1 Обзор предметной области

Связи пользователей социальной сети естественным образом представляются с помощью графовой модели, где вершинами являются пользователи, а ребрами — связи между ними (например, «дружба» и т.д.). Пригодное для анализа представление должно удовлетворять следующим критериям:

— Минимимальность пересечений ребер;

— Равномерность распределение вершин;

— Однородность длин ребер;

— Наличие симметрии.

Силовые алгоритмы визуализации графов представляют собой класс алгоритмов, цель которых заключается в улучшении эстетических характеристик графического представления. Основной задачей этих алгоритмов является такое размещение узлов графа в двухмерном или трехмерном пространстве, чтобы длины рёбер были приблизительно одинаковыми, и при этом количество пересечений рёбер было минимальным. Пример визуализации графа связей статей на Wikipedia, полученной с помощью силового алгоритма изображен на рисунке 1

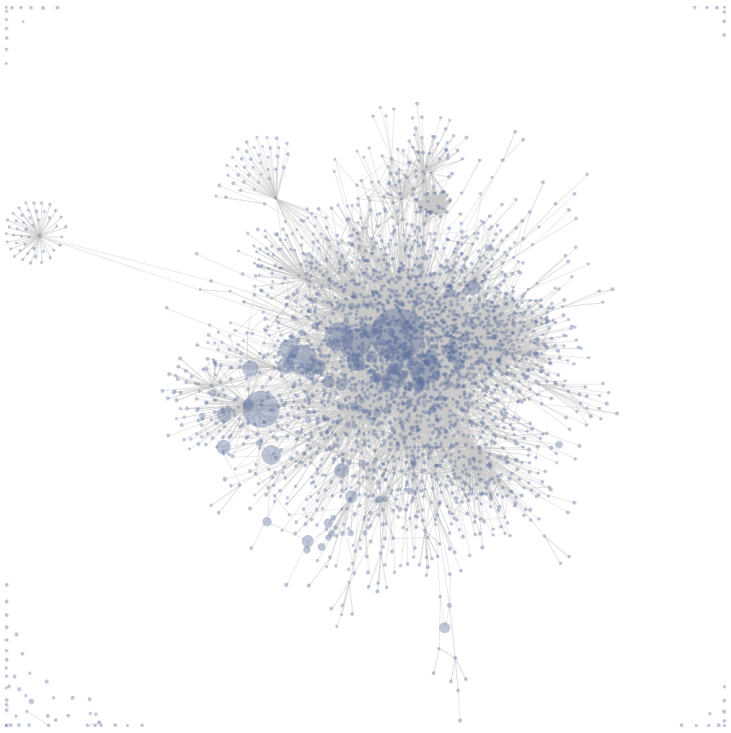


Рисунок 1 — Визуализация графа с помощью силового алгоритма.

1.1 Силовые алгоритмы

Силовые алгоритмы визуализации графов оперируют основными принципами сил и энергии в физическом смысле, чтобы достичь оптимального распределения узлов и рёбер в графе.

Основные этапы и принципы их работы:

1. Инициализация. Начальное распределение узлов графа задаётся случайным образом или с использованием предварительных координат;
2. Определение сил. Алгоритм вычисляет силы для каждого узла, основываясь на их текущих относительных положениях.
3. Обновление координат. На основе вычисленных сил происходит обновление координат узлов.
4. Итерация. Процесс вычисления сил и обновления координат повторяется в циклах, обычно до тех пор, пока не будет достигнута определенная степень стабилизации или заданное количество итераций.

Несмотря на широкое применение, силовые алгоритмы визуализации графов также имеют свои ограничения и минусы:

1. Чувствительность к начальным условиям: Результаты силовых алгоритмов могут зависеть от начального распределения узлов. Различные начальные условия могут привести к разным конечным визуализациям, что может затруднить воспроизводимость результатов.
2. Зависимость от параметров: Силовые алгоритмы имеют различные параметры, такие как коэффициенты отталкивания и притяжения. Оптимальные значения этих параметров могут зависеть от конкретного графа, и их настройка может потребовать экспериментов.
3. Проблемы с большими графами: При визуализации очень больших графов силовые алгоритмы могут потребовать значительных вычислительных ресурсов и времени. Это может сделать их неэффективными для работы с масштабными социальными сетями или другими крупными графовыми структурами.

Силовые алгоритмы визуализации графов являются подходящим инструментом для задачи визуализации графа связей пользователей социальной сети. Несмотря на некоторые ограничения, такие как чувствительность к

начальным условиям и зависимость от параметров, эти алгоритмы обладают важными преимуществами.

Естественное моделирование физических свойств позволяет отразить реальные взаимодействия в социальных сетях. Кроме того, силовые алгоритмы обеспечивают эстетически привлекательные визуализации с равномерным распределением вершин и минимизацией пересечений рёбер.

1.2 Алгоритм Идеса

Алгоритм визуализации графов, разработанный Питером Идесом, представляет собой метод для распределения вершин графа на плоскости таким образом, чтобы минимизировать пересечения рёбер и создать визуально удовлетворительное представление структуры графа. Этот алгоритм был представлен в 1984 году [1].

Каждое ребро графа моделируется как пружина, которая стремится удерживать связанные вершины на определенном расстоянии друг от друга. Если расстояние между вершинами больше желаемого, то действует пружинная сила, направленная на уменьшение расстояния. Если расстояние меньше, то действует отталкивающая сила, направленная на увеличение расстояния. Вершины, не соединенные ребрами, действуют друг на друга отталкивающим образом. Это содействует распределению вершин по плоскости графа, предотвращая их слишком близкое расположение.

При использовании алгоритма Идеса необходимо определить следующие константы:

1. Число итераций. Это количество итераций, которые алгоритм будет выполнять для уточнения распределения вершин. Большее число итераций может привести к более точной визуализации, но также увеличит вычислительную сложность.
2. Коэффициент жесткоски пружин. Этот коэффициент определяет силу, с которой пружины стремятся удерживать соединенные вершины на определенном расстоянии. Увеличение этого коэффициента сделает пружины более жесткими, а уменьшение — более мягкими.
3. Коэффициент отталкивания. Определяет силу отталкивания между вершинами, которые не соединены ребрами. Увеличение этого

коэффициента усиливает силы отталкивания, что приводит к более широкому распределению вершин.

4. Длина пружины. Задает желаемую длину пружины между соединенными вершинами. Этот параметр влияет на силы пружин и, таким образом, на расстояние между вершинами.
5. Шаг обновления. Это значение определяет величину изменения позиции вершин на каждой итерации. Большие значения могут привести к быстрому перемещению вершин, но могут также вызвать колебания и неустойчивость, тогда как малые значения могут сделать процесс слишком медленным.

Псевдокод алгоритма Идеса представлен на рисунке 2

Листинг 1 Алгоритм Идеса

function $f_{spring}(p_u, p_v)$

$r \leftarrow c_{spring} \log \frac{\|p_v - p_u\|}{l} \cdot \overrightarrow{p_u p_v}$

return r

function $f_{rep}(p_u, p_v)$

$r \leftarrow \frac{c_{rep}}{\|p_v - p_u\|} \cdot \overrightarrow{p_u p_v}$

return r

function Eades($G = (V, E)$, $p = (p_v)_{v \in V}$, $k \in \mathbb{N}$)

$t \leftarrow 1$

while $t \leq K$ **do**

for $u \in V$ **do**

$F_u(t) \leftarrow \sum_{v:\{u,v\} \notin E} f_{rep}(u, v) + \sum_{v:\{u,v\} \in E} f_{spring}(u, v)$

for $u \in V$ **do**

$p_u \leftarrow p_u + \delta \cdot F_u(t)$

$t \leftarrow t + 1$

return p

Рисунок 2 — Псевдокод алгоритма Идеса

1.3 Алгоритм Фрюхтермана-Рейнгольда

Алгоритм Фрюхтермана-Рейнгольда был предложен Томасом Фрюхтерманом и Эдвардом Рейнгольдом в 1991 году [2]. Этот алгоритм рассматривает вершины графа как «частицы», оказывающие друг на друга притягивающие и отталкивающие силы. Притягивающие и отталкивающие силы переопределяются формулами 1 и 2 соответственно в терминах расстояния d между двумя вершинами и оптимального расстояния между вершинами, которое определяется формулой 3

$$f_{attr}(d) = \frac{d^2}{k} \tag{1}$$

$$f_{rep}(d) = -\frac{k^2}{d} \tag{2}$$

$$k = C\sqrt{\frac{width * heigth}{|V|}} \tag{3}$$

Этот алгоритм похож на алгоритм Идеса в том, что оба алгоритма вычисляют притягивающие силы между соседними вершинами и отталкивающие силы между всеми парами вершин. Алгоритм Фрюхтермана и Рейнгольда добавляет понятие «температура», которое можно использовать следующим образом: «температура» может начинаться с начального значения (например, с одной десятой ширины рамки) и убывать до 0 обратным линейным образом. Температура контролирует смещение вершин, так что по мере улучшения компоновки корректировки становятся все меньше. Использование температуры здесь является частным случаем общей техники, называемой имитационным отжигом.

Псевдокод алгоритма Фрюхтермана и Рейнгольда показан на рисунке 3. На каждой итерации базовый алгоритм вычисляет $O(|E|)$ притягивающих сил и $O(|V|^2)$ отталкивающих сил. Чтобы уменьшить квадратичную сложность вычисления отталкивающих сил, Фрюхтерман и Рейнгольд предлагают использовать сеточный вариант своего базового алгоритма, в котором отталкивающие силы между удаленными вершинами игнорируются. Для разреженных графов и при равномерном распределении вершин этот метод позволяет приближенно вычислить силы отталкивания за время $O(|V|)$.

Листинг 2 Алгоритм Фрюхтермана-Рейнгольда

function $f_{rep}(p_u, p_v)$

$r \leftarrow \frac{l^2}{\|p_v - p_u\|} \cdot \overrightarrow{p_v p_u}$

return r

function $f_{attr}(p_u, p_v)$

$r \leftarrow \frac{\|p_v - p_u\|^2}{l} \cdot \overrightarrow{p_u p_v}$

return r

function FruchtermanReingold($G = (V, E)$, $p = (p_v)_{v \in V}$, $k \in \mathbb{N}$)

$t \leftarrow 1$

while $k \leq K$ **do**

for $u \in V$ **do**

$F_u(k) \leftarrow \sum_{v \in V} f_{rep}(u, v) + \sum_{v:\{u,v\} \in E} f_{attr}(u, v)$

for $u \in V$ **do**

$p_u \leftarrow p_u + \delta(t) \cdot F_u(k)$

$t \leftarrow cool(t)$

$k \leftarrow k + 1$

return p

Рисунок 3 — Псевдокод алгоритма Фрюхтермана-Рейнгольда

Основные шаги алгоритма:

1. Инициализация расположения вершин. Вершины графа случайным образом размещаются в двумерном пространстве;
2. Вычисление отталкивающих сил. Для каждой пары вершин вычисляется отталкивающая сила на основе закона Кулона;
3. Вычисление притягивающих сил. Для каждой пары вершин вычисляется притягивающая сила на основе закона Гука;
4. Обновление расположения вершин. На основе вычисленных сил обновляются координаты вершин. Вершины двигаются в направлении суммарной силы;

5. Уменьшение температуры. Уменьшение температуры позволяет системе постепенно переходить от хаотичного состояния к устойчивому распределению.

1.4 Алгоритм Камады-Кавай

Алгоритм Камады-Кавай[3] представляет собой метод визуализации графов, разработанный в 1989 одноименными японскими учеными. Этот метод основан на энергетической модели, где вершины графа представляются как частицы, а рёбра — как пружины. Цель алгоритма — подобрать такое положение вершин, чтобы евклидово расстояние между ними соответствовало расстоянию в графе, т.е минимальному длину пути.

В этой модели «идеальным» рисунком графа будет тот, в котором парные евклидовы расстояния между нарисованными вершинами совпадают с парными теоретическими расстояниями графа, вычисленными в результате работы алгоритма нахождения кратчайшего пути по всем парам, например, с помощью алгоритма Флойда Уоршела [4]. Поскольку эта цель не всегда может быть достигнута для произвольных графов в двумерном или трехмерном евклидовом пространстве, подход основан на настройке системы пружин таким образом, чтобы минимизация энергии системы соответствовала минимизации разницы между геометрическими и графовыми расстояниями. В этой модели нет отдельных притягивающих и отталкивающих сил между парами вершин, но вместо этого, если пара вершин находится (геометрически) ближе/дальше, чем соответствующее расстояние на графе, вершины отталкиваются/притягиваются друг к другу. Пусть d[i][j] обозначает расстояние кратчайшего пути между вершиной i и вершиной j в графе. Тогда идеальная длина пружины между вершинами i и j вычисляется как произведение желательной длины одного ребра L на величину d[i][j]. Камада и Каваи предлагают в качестве желательной длины ребра использовать величину равную отношению длины стороны экрана к максимальному длину пути, среди всех кратчайших путей в графе. Сила пружины между вершинами i и j определяется как отношение некоторой заранее заданной константы к квадрату длины пути между вершинами. Рассматривая задачу рисования как локализацию n частиц в двумерном евклидовом пространстве, можно использовать следующую функцию 4 общей энергии

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{i,j} (|p_i - p_j| - l_{i,j})^2 \tag{4}$$

Координаты частицы в двумерном евклидовом пространстве задаются парой чисел, поэтому можно переписать функцию энергии следующим образом:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{i,j} \left((x_i - x_j)^2 + (y_i - y_j)^2 + l_{i,j}^2 - 2 l_{i,j} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right) \tag{5}$$

Цель алгоритма — найти значения переменных, которые минимизируют функцию энергии E, зависящую от 2n переменных, где n — количество вершин графа. В частности, в локальном минимуме все частные производные равны нулю, что соответствует решению 2n одновременных нелинейных уравнений. Поэтому Камада и Каваи вычисляют устойчивое положение по одной частице за раз. Рассматривая E как функцию только xm и ym, можно вычислить минимум E используя метод Ньютона-Рафсона. На каждом шаге алгоритма выбирается частица под номером m с наибольшим значением величины опередяемой формулой 6.

$$\Delta_m = \sqrt{\left(\frac{\partial E}{\partial x_m} \right) + \left(\frac{\partial E}{\partial y_m} \right)} \tag{6}$$

Алгоритм Камады и Каваи требует больших вычислительных затрат, т.к. небходимо подсчет кратчайших путей между всеми парами вершин, которое можно выполнить за время $O(|V|^3)$ с помощью алгоритма Флойда-Уоршалла или за $O(|V|^2 log(|V|) + |E||V|)$ с помощью алгоритма Джонсона. Более того, этот алгоритм требует $O(|V|^2)$ памяти для хранения парных расстояний между вершинами. Несмотря на более высокую временную и пространственную сложность, алгоритм дает простое и интуитивно понятное определение «хорошей» компоновки графа: компановка графа является хорошей, если геометрические расстояния между вершинами соответствуют расстояниям между вершинами, лежащим в основе графа.

Псевдокод алгоритма Камады и Каваи представлен на листинге 4.

Листинг 3 Алгоритм Камада-Кавай

function KamadaKawai($G = (V, E)$, ε, $p = (p_v)_{v \in V}$, K)

$d \leftarrow FloydWarshall(G)$

for $i \leq |V|$ **do**

for $j \leq |V|$ **do**

if $i \neq j$ **then**

$l_{i,j} \leftarrow d_{i,j} \times \frac{L_0}{max_{i_1 < j_1}(d_{i_1,j_1})}$

$k_{i,j} \leftarrow \frac{K}{d_{i,j}^2}$

while $max_i \Delta_i >$ ε **do**

$\Delta_m \leftarrow max_i \Delta_i$

while $\Delta_m >$ ε **do**

Вычислить $\delta x, \delta y$ решив следующую систему

$\frac{\partial^2 E}{\partial x_m^2}(x_m, y_m)\delta x + \frac{\partial^2 E}{\partial x_m \partial y_m}(x_m, y_m)\delta y = -\frac{\partial E}{\partial x_m}(x_m, y_m)$

$\frac{\partial^2 E}{\partial y_m \partial x_m}(x_m, y_m)\delta x + \frac{\partial^2 E}{\partial y_m^2}(x_m, y_m)\delta y = -\frac{\partial E}{\partial y_m}(x_m, y_m)$

$p_m.x = p_m.x + \delta x$

$p_m.y = p_m.y + \delta y$

return p

Рисунок 4 — Псевдокод алгоритма Камады-Кавай

2 Разработка приложения

Приложение будет являться клиентской библиотекой, предостовляющей возможности отрисовки графов с помощью нескольких силовых алгоритмов, рассмотренных выше. Кроме функционала для отрисовки, необходимо разработать интерфейсы и модели для взаимодействия с модулем для работы с API социальных сетей. Таким образом, станет возможно отделить логику визуализации от логики получения и преобразования данных пользователей и их связей.

2.1 Архитектура приложения

1. Модуль сбора данных. Этот модуль представляет собой первый этап обработки данных в библиотеке и отвечает за сбор информации из социальной сети. Данный модуль является заменяемым компонентом, что позволяет интегрировать уникальные методы сбора данных для различных социальных сетей, сохраняя при этом совместимость с остальными модулями. Задачи модуля включают:
    - Получение данных: модуль осуществляет запросы к API социальной сети для получения информации о пользователях, связях между ними и других существенных данных;
    - Преобразование данных: полученная информация преобразуется в формат, легко обрабатываемый другими частями приложения. Это может включать в себя фильтрацию ненужной информации и преобразование их в структуры, более удобные для последующей обработки.
2. Модуль обработки данных. Этот модуль выполняет обработку данных, полученных от модуля сбора, и преобразует их в графовую модель. Задачи этого модуля включают:
    - Формирование графовой структуры: модуль создает структуры данных, представляющие собой граф с определенными координатами вершин. Эти данные готовы к передаче модулю визуализации;
    - Применение силовых алгоритмов: К модулю поступают данные о вершинах графа, и на них применяются силовые алгоритмы.

3. Модуль визуализации занимается отображением графа, обработанного силовым алгоритмом. Его задачи включают:
    — Отображение вершин и рёбер: Модуль визуализации отображает граф, используя координаты вершин и информацию о связях между ними. Это включает в себя определение визуального представления вершин и рёбер;
    — Интерактивность: Предоставление пользователю возможности взаимодействия с визуализацией. Это может включать в себя приближение/удаление, перемещение вершин, отображение дополнительной информации при наведении и т. д.

Эти три модуля в совокупности обеспечивают работу от сбора данных из социальной сети до визуализации обработанного графа с применением силовых алгоритмов.

Схема работы приложения представлена на рисунке 5

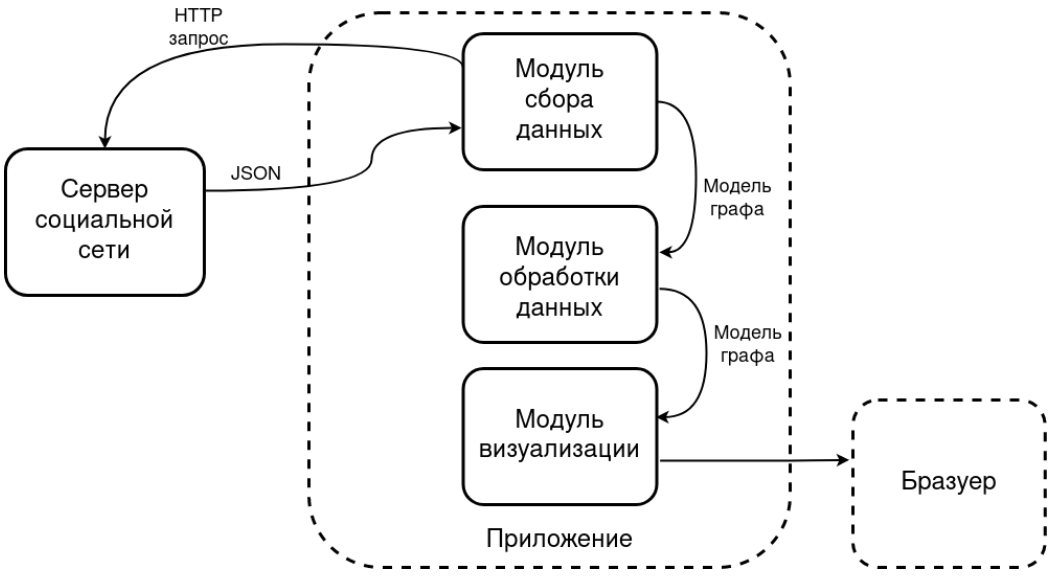


Рисунок 5 — Схема взаимодействия модулей приложения.

2.2 Выбор инструментов разработки

Для разработки библиотеки был выбран язык TypeScript [5] из-за следующих его особенностей:

1. Статическая Типизация. TypeScript предлагает статическую типизацию, что позволяет выявлять и предотвращать множество ошибок на

этапе разработки, что в свою очередь способствует более безопасной разработке.

2. Расширенная Поддержка ООП. TypeScript, как расширение JavaScript, обладает мощными возможностями объектно-ориентированного программирования. Это особенно важно при создании таких структур данных как графы. Классы и интерфейсы TypeScript облегчают организацию кода и создание модульных и поддерживаемых компонентов.
3. Экосистема JavaScript. TypeScript является надмножеством JavaScript, что означает полную совместимость с существующей JavaScript-экосистемой. Это позволяет использовать существующие библиотеки и инструменты в разработке библиотеки, обеспечивая гибкость и расширяемость проекта.
4. Поддержка Современных Стандартов. TypeScript активно поддерживается и обновляется, включая поддержку последних стандартов ECMAScript. Это важно для использования новых возможностей языка.
5. Инструменты Для Рефакторинга и Анализа Кода. TypeScript обеспечивает богатый набор инструментов для рефакторинга кода и анализа его качества. Это упрощает процессы поддержки и развития библиотеки.

После выборая ЯП, следующим шагом является выбор подходящей библиотеки визуализации. Библиотеки для визуализации предоставляют ряд инструментов, необходимых для эффективного представления и взаимодействия с данными графов. Среди всех продуктов больше всего выделяются: D3.js, Cytoscape.js, Vis.js. Среди всех перечисленных вариантов для данного проекта лучше всего подходит Cytoscape.js [6], т.к. специализируется именно на визуализации графов, что делает её более оптимизированной для конкретно этого вида задач. Кроме того, важны следующие факторы:

1. Cytoscape.js предоставляет обширный функционал для работы с графами, включая поддержку различных макетов, стилей, анимаций и обработки событий. Это обеспечивает гибкость в адаптации библиотеки под уникальные требования проекта;

2. Библиотека Cytoscape.js оптимизирована для эффективной отрисовки крупных графов, что соответствует требованиям проекта по визуализации сложных структур данных. Оптимизированные алгоритмы позволяют обеспечить высокую производительность даже при работе с большим количеством элементов;
3. Cytoscape.js находится под активной поддержкой сообщества разработчиков. Регулярные обновления и внимание к запросам сообщества гарантируют наличие актуальных версий библиотеки и устранение возможных проблем.

2.3 Разработка моделей приложения

Для эффективной работы с данными социальных сетей в приложении необходимо разработать две ключевые модели — модель пользователя и модель графа. Эти модели будут играть центральную роль в преобразовании и структурировании данных для последующей визуализации и взаимодействия.

Модель пользователя представляет абстракцию данных о каждом участнике социальной сети. Сервер передает данные в формате JSON [7], который необходимо преобразовать в объекты языка программирования для дальнейшего удобства взаимодействия.

Поля модели поользователя:

— Идентификатор: уникальный идентификатор пользователя в социальной сети;
— Имя и Фамилия: информация о имени и фамилии пользователя;
— Фотография: ссылка на фотографию пользователя.

Модель пользователя служит для:

— Преобразования данных из формата JSON в объекты языка программирования;
— Хранения основных свойств пользователя для дальнейшего использования в приложении.

Модель графа необходима для структурирования информации о связях между пользователями и предоставления основы для визуализации графа социальной сети. Модель графа включает в себя модель вершины, которая является расширением модели пользователя.

Объекты модели графа:

— Модель вершины является расширенной моделью пользователя, содержащей дополнительную информацию о расположении пользователя в графе;
— Список инцидентности: Информация о взаимосвязях между пользователями.

Модель графа выполняет следующие функции:

— Организация и хранение связей между пользователями;
— Возможность определения расположения каждого пользователя в графе;
— Структурирование данных для визуализации и взаимодействия с графом социальной сети.

Разработанные модели обеспечивают необходимый фундамент для эффективной работы с данными социальных сетей. Модель пользователя обеспечивает удобное представление информации о каждом участнике сети, в то время как модель графа структурирует и предоставляет основу для визуализации связей между пользователями.

2.4 Разработка силовых алгоритмов

Для использования в разрабатываемой библиотеке были выбраны алгоритмы Фрюхтермана-Рейнгольда и Камады-Кавай. Алгоритма Идеса, хотя и эффективен для небольших графов, но уступает выбранным алгоритмам в практической применимости. Алгоритм Идеса имеет скорее теоретическую ценность, так как он является одним из первых силовых алгоритмов, на основе которого развиваются более совершенные методы в данной области.

Указанные алгоритмы оперируют двумерными векторами, поэтому в первую очередь небходимо разработать класс двумерного вектора со следующими операциями:

— сложение
— вычитание
— умножение на скаляр
— получение евклидовой нормы
— нормализация

3 Реализация приложения

Для выполнения поставленной задачи необходимо реализовать три модуля приложения отвечающих за сбор, обработку и пребразование данных социальной сети.

Разработка проекта осуществлялась в редакторе кода Neovim [8]. Данный редактор может быть использван в качестве IDE благодаря обширной библиотеке расширений. Расширение «typescript-language-server» [9] предоставляет умное автодополнение, инструменты для рефакторинга и анализа кода — возможности, необходимые для быстрой и эффективной разработки на typescript.

Для сборки проекта используется инструмент сборки для веб-приложений под названием Parcel [10]. Он предоставляет простой и быстрый способ управления зависимостями и создания оптимизированных пакетов кода для развертывания веб-приложений.

3.1 Особенности реализации модуля сбора данных

Для сбора данных была выбрана социальная сеть «Вконтакте». Данная соцсеть предоставляет публичный API [11], с помощью которого можно получить данные о пользователях и их связях. Для обращения к API был реализован отдельный модуль, содержащий следующие компоненты:

— Класс VkAPI, предоставляющий методы для получения всех друзей определенного пользователя, и метод для получение связей внутри некоторой группы пользователей(реализация класса представлен на листинге 3);

— Тип данных User, небходимый для преобразование ответа из формата JSON в объект языка(реализация представлена на листинге 1);

— Набор типов представляющих собой структуры ответов на запросы к API(реализация представлена на листинге 2)

1. Тип ErrorResponse — содержит сообщение об ошибке в случае запроса, который завершился неуспешно.
2. Тип GetFriendsResponse — представляет собой успешный ответ на запрос всех друзей пользователя. Содержит поле,

указывающие на количество друзей, и массив со структурами User, описывающий всех друзей запрашиваемого пользователя.

3. Тип GetMutualResponse — представляет собой успешный ответ на запрос о связях внутри группы. Данный тип является массивом объектов, которые описывают какие связи имеет каждый пользователь группы.

3.2 Особенности реализации алгоритма Фрюхтермана-Рейнгольда

Для реализации функциональность алгоритма Фрюхтермана-Рейнгольдаа был написан отдельный модуль, который состоит из двух локальных и одной экспортируемой функций , каждая из которых выполняет свою задачу в рамках алгоритма.

— Функция подсчета отталкивающей силы (представлена на листинге 5);
— Функция подсчета притягивающей силы (представлена на листинге 6);
— Функция реализующая главный алгоритм (представлена на листинге 4). Она управляет взаимодействием отталкивающих и притягивающих сил, а также обновляет координаты узлов для достижения оптимального распределения;

Данный модуль предоставляет высокоуровневый интерфейс, с помощью которого можно преобразовать граф согласно алгоритму Фрюхтермана-Рейнгольда. Локальные функции (подсчет отталкивающей и притягивающей сил) не видны из других модулей, что обеспечивает инкапсуляцию и изоляцию внутренних деталей алгоритма.

3.3 Особенности реализации алгоритма Камады-Кавай

Посколько алгоритм Камады-Кавай является более сложным в реализации, для него был написан отдельный класс, предоставляющий методы для преобразования графа. Для реализации данного алгоритма были реализованы следующие функции и методы:

— Алгоритм Флойда-Уоршела для поиска кратчайших путей в графе (представлен на листинге 7);
— Метод поиска значения энергии вершины (представлен на листинге 8);
— Метод поиска вершины с максимальным значением энергии (представлен на листинге 9);
— Метод вычисления нового положения вершины (представлен на листинге 10);
— Главный метод, реализующий основной алгоритм (представлен на листинге 11);

Для применения к графу алгоритма Камады-Кавай небходимо создать объект KamadaKawai, передав в конструктор граф, к которому должен применяться алгоритм. Далее, ипользуя вызвов метода run, можно получить граф с оптимальной компановкой.

3.4 Описание работы приложения

Для работы с модулем сбора данных социальной сети «Вконтакте» небходимо получить специальный ключ, который будет отправляться с каждый обращением к API. Если данный ключ получен, то работа приложения строится следущим образом:

1. Небходимо создать объкт VkAPI, передав в конструктор полученный ранее ключ;
2. Для рассмотрения связей внутри группы друзей некоторого пользователя небходим получить его уникальный цифрофой идентификатор.
3. Поскольку данное приложение полностью выполняется в браузере, то при запросе к API социальной сети «Вконтакте» возникают проблемы связанные с CORS [12]. Для решения данных проблем небходимо воспользоваться протоколом JSONP [13], согласно котору для выполнения запроса нужно создать элемент script с свойством src равным тому адресу, к которому небходимо совершить запрос. Также данный адрес должен содержать название функции обратного вызова, которая будет вызвана в момент возвращения ответа. Каждый из методов для запроса данных объекта VkAPI содержит параметр, который является функцией, которая должна будет вызваться при возвращение ответа.

4. После получения всех данных небходимо сконструировать объект графа, передав в конструктор массив пользователей графа и матрицу смежности.
5. К полученному графу небходимо применить один из доступных алгоритмов, и передать преобразованный граф в функцию draw(представлена на листинге 12), вместе с предварительно созданным объектом типа cytoscape.Core()

Запуск приложения производиться согласно рисунку 6. После успешной сборки в консоли появится адрес, а также откроется окно браузера с визуализацией графа. Примеры визуализации графа из 60 пользователей с помощью алгоритмов Фрюхтермана-Рейнгольда и Камады-Кавай представлены на рисунках 7 и 8 соответственно.

```
❯ npx parcel serve index.html --open firefox
Server running at http://localhost:1234
✨ Built in 429ms
```

Рисунок 6 — Пример запуска приложения.

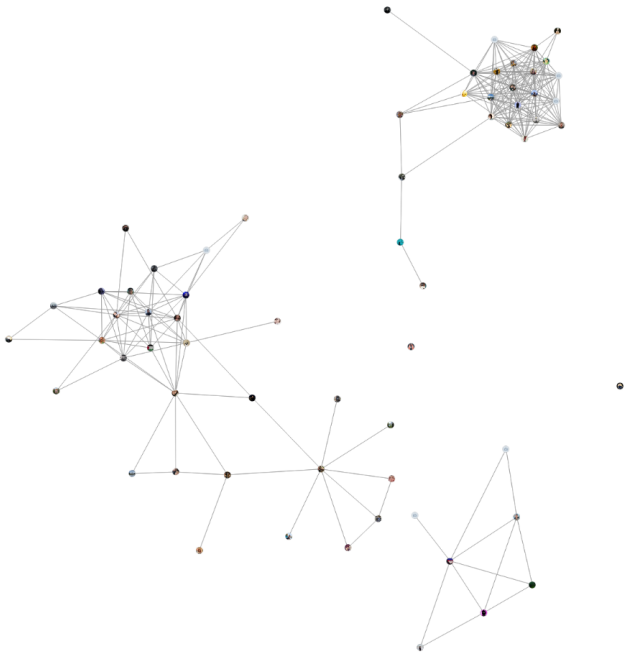


Рисунок 7 — Результат визуализации с помощь алгоритма Фрюхтермана-Рейнгольда.

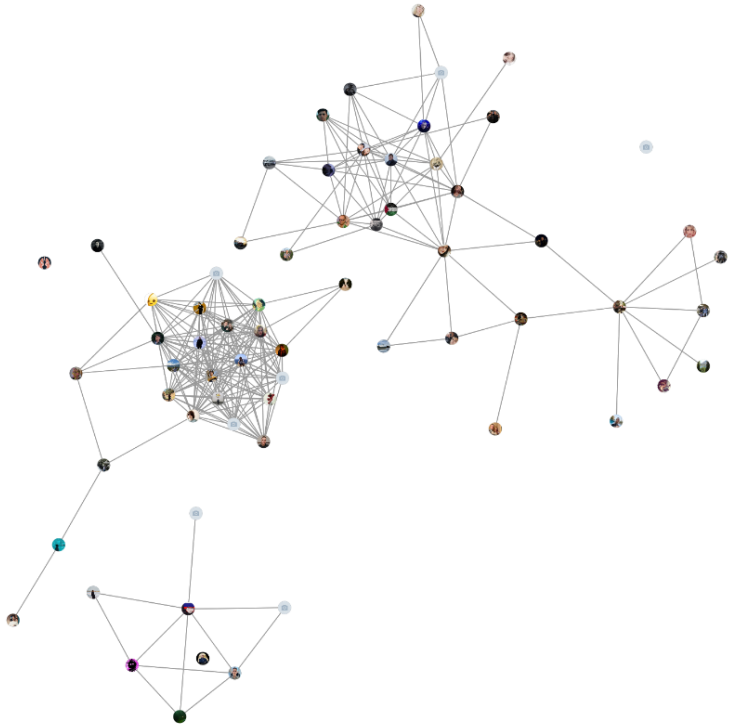


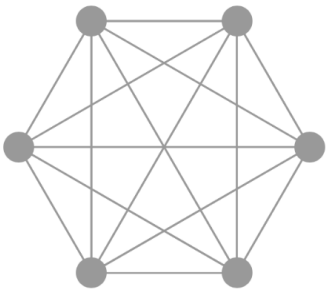
Рисунок 8 — Результат визуализации с помощью алгоритма Камады-Кавай.

4 Тестирование

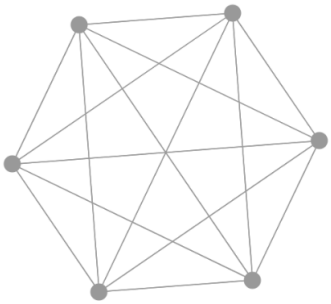
Для тестирования приложения проведем сравнения работы программы, с результатми полученными программой graphviz [14]. Сравнение будет производится на различных, с точки зрения структуры, графах.

4.1 Тестирование на полном графе

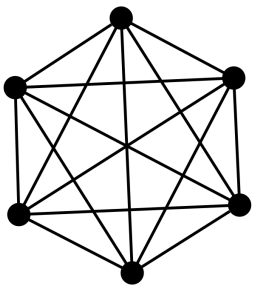
Результат работы программы на небольшом полном графе представлен на рисунке 9, пункты а и б. Результат визуализации с помощью graphviz представлен на том же рисунке, пункт в. Как можно увидеть, укладки графов идентичны, с точность до поворота.



а) алгоритм Фрюхтермана-Рейнгольда.



б) алгоритм Камады-Кавай.

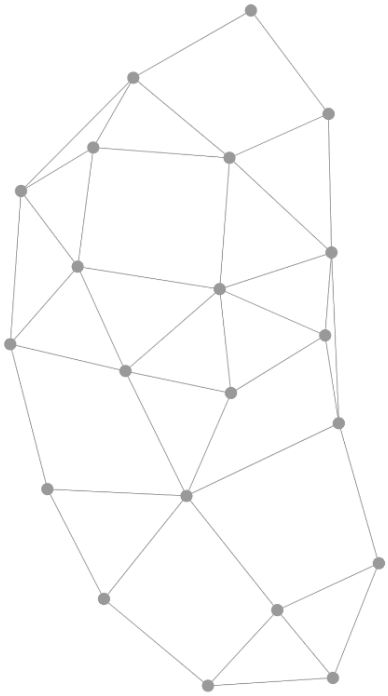


в) программа Graphviz.

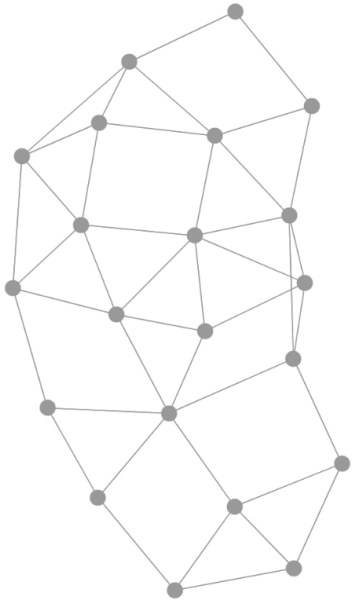
Рисунок 9 — Результат работы программы на полном графе.

4.2 Тестирование на плотном графе

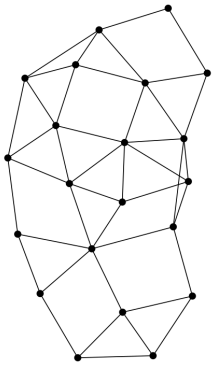
Результат работы программы на небольшом плотном графе представлен на рисунке 10, пункты а и б. Результат визуализации с помощью graphviz представлен на том же рисунке, пункт в. Результаты визуализации практически идентичны.



а) алгоритм Фрюхтермана-Рейнгольда.



б) алгоритм Камады-Кавай.



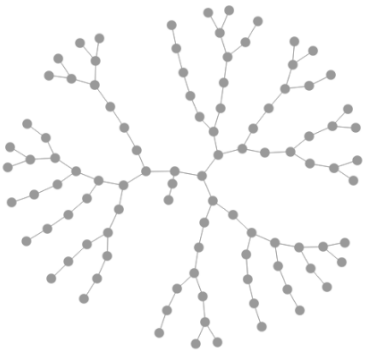
в) программа Graphviz.

Рисунок 10 — Результат работы программы на небольшом плотном графе.

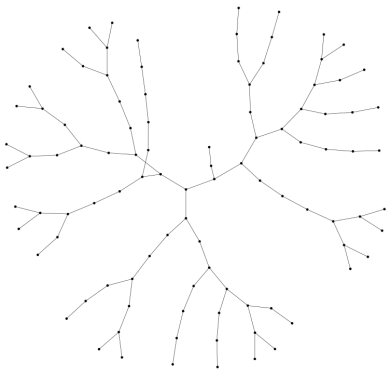
4.3 Тестирование на бинарном дереве

Результат работы программы на графе, не содержащем циклов, и каждая вершина которого имеет не более трех смежных ребер, представлен на рисунке 11, пункты а и б. Как можно увидеть, результататы полученные разработанной программой, и с помощью graphviz, практически совпадают.

а) алгоритм Фрюхтермана-Рейнгольда.



б) алгоритм Камады-Кавай.

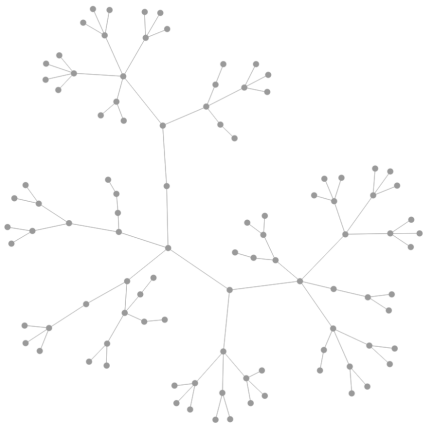


в) программа Graphviz.

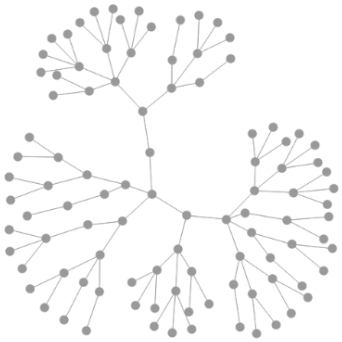
Рисунок 11 — Результат работы программы на бинарном дереве.

4.4 Тестирование на k-арном дереве

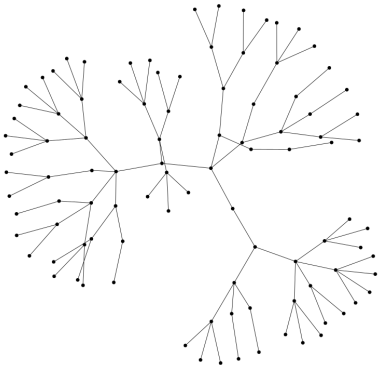
Результат работы программы на графе, являющимся k-арным деревом при k = 4, представлен на рисунке 12, пункты а и б. Как можно увидеть, результататы полученные разработанной программой, и с помощью graphviz, практически совпадают.



а) алгоритм Фрюхтермана-Рейнгольда.



б) алгоритм Камады-Кавай.



в) программа Graphviz.

Рисунок 12 — Результат работы программы на k-арном дереве, при k=4.

4.5 Результаты тестирования

В результате тестирования программы на различных структурах графов и сравнения её результатов с визуализацией, полученной с использованием Graphviz, можно утверждать о правильности работы программы. Программа продемонстрировала согласованность и точность в создании расположений для разнообразных типов графов, включая полные, плотные и графы с ограничением на количество смежных рёбер у вершин. На каждом этапе тестирования результаты работы программы оказывались практически идентичными визуализации, полученной с помощью Graphviz.

Обнаруженные расхождения между результатами программы и Graphviz были минимальными и, в большинстве случаев, могут быть объяснены допустимыми вариациями в укладке графов.

Таким образом, результаты тестирования позволяют сделать вывод о правильности работы разработанной программы и её способности генерировать расположения графов, имеющих различные характеристики.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был проведен анализ способов визуализации связей пользователей в социальных сетях. Рассмотрены различные алгоритмы для визуализации графовых данных с учетом особенностей структуры социальных связей.

Основной целью данной курсовой работы была реализация программы для визуализации связей пользователей в социальной сети. Результаты выполнения задачи демонстрируют удовлетворительное качество работы программы, что подтверждается сравнением с результатами визуализации, полученными с использованием Graphviz. Разработанное программное решение представляет собой инструмент для анализа и изучения структуры социальных связей в различных сообществах.

В заключение, несмотря на достигнутые положительные результаты, существует потенциал для дальнейшего улучшения программы. Это включает в себя оптимизацию механизмов обработки и анализа данных, адаптацию программы для различных типов социальных сетей, а также повышение удобства использования в различных сценариях исследования социальных взаимосвязей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Eades P. A heuristic for graph drawing // Congressus numerantium. — 1984. — С. 149—160. — (дата обращения: 26.11.2023).
2. Fruchterman T., Reingold E. Graph Drawing by Force-directed Placement // Software - Practice and Experience. — 1991. — С. 1129—1164. — (дата обращения: 28.11.2023).
3. Kamada T., Kawai S. An algorithm for drawing general undirected graphs // Information Processing Letters. — 1989. — С. 7—15. — (дата обращения: 10.12.2023).
4. Алгоритм Флойда. — URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A4%D0%BB%D0%BE%D0%B9%D0%B4%D0%B0 ; (дата обращения: 28.12.2023).
5. Документация TypesScript. — URL: https://www.typescriptlang.org/docs/ ; (дата обращения: 18.01.2024).
6. Документация библиотки Cytoscape.js. — URL: https://js.cytoscape.org ; (дата обращения: 18.12.2023).
7. Документация JSON. — URL: https://www.json.org/json-en.html ; (дата обращения: 15.12.2023).
8. Редактор кода neovim. — URL: https://neovim.io/ ; (дата обращения: 18.01.2024).
9. Языковой сервер для typescript. — URL: https://github.com/typescript-language-server/typescript-language-server ; (дата обращения: 18.01.2024).
10. Документация Parcel. — URL: https://parceljs.org/docs/ ; (дата обращения: 18.01.2024).
11. Документация VK API. — URL: https://dev.vk.com/ru/api/overview ; (дата обращения: 15.12.2023).
12. Механизм CORS. — URL: https://developer.mozilla.org/ru/docs/Web/HTTP/CORS ; (дата обращения: 18.01.2024).

13. Описание протокола JSONP. — URL: https://learn.javascript.ru/ajax-jsonp ; (дата обращения: 18.01.2024).

14. Визуализатор графов Graphviz. — URL: https://graphviz.org/ ; (дата обращения: 18.01.2024).

ПРИЛОЖЕНИЕ А

Листинг 1: Модель пользователя

```
export type User = {
  id: string;
  first_name: string;
  last_name: string;
  photo_100: string;
};
```

Листинг 2: Структуры ответов

```
import { User } from "./user";

export type ApiResponse = {
  response: GetMutualResponse | GetFriendsResponse;
} & ErrorResponse;

export type ErrorResponse = {
  error: string;
};

export type GetFriendsResponse = {
  count: Number;
  items: User[];
};

export type GetMutualResponse = {
  common_count: Number;
  common_friends: Number;
  id: string;
}[];
```

Листинг 3: Реализация класса VkAPI

```
const API_URL_PREFIX = "https://api.vk.com/method/";
const API_VERSION = "5.154";
const API_URL_SUFFIX = `&v=${API_VERSION}`;

type FriendsRequestParam = {
  key: string;
  value: string;
};

export class VkAPI {
  token: string;
  constructor(token: string) {
    this.token = token;
  }

  public getFriends(
    id: string,
    params: FriendsRequestParam[],
    callback: Function
  ) {
    let url = API_URL_PREFIX + `friends.get?user_id=${id}`;

    params.forEach(({ key, value }) => {
      url += `&${key}=${value}`;
    });

    this.makeApiRequest(url, "getFriends", callback);
  }

  public getMutual(sourceUid: string, users: User[], callback: Function)
↪   {
    let url =
      API_URL_PREFIX +

↪   `friends.getMutual?source_uid=${sourceUid}&target_uids=${users.reduce(
        (acc, user) => {
          return acc + user.id + ",";
        },
        ""
      )}`;
    this.makeApiRequest(url, "getMutual", callback);
  }
}
```

Листинг 4: Реализация алгоритма Фрюхтермана-Рейнгольда

```
export const FruchtermanReingold = ( graph: Graph, k: number, iterCount:
↪  number) => {
  graph = makeCircle(graph);
  let mp: { [_: string]: Vector } = {};
  let t = 10 * Math.sqrt(Object.keys(graph.vertices).length);
  for (let l = 0; l < iterCount; l++) {
    for (const u in graph.vertices) {
      mp[u] = new Vector(0, 0);
    }
    const set = new Set();
    for (const u in graph.vertices) {
      for (const v in graph.vertices) {
        if (u == v || set.has([u, v] || set.has([v, u]))) continue;

        const [force, dist] = findRepulsive(
          Vector.fromVertex(graph.vertices[v]),
          Vector.fromVertex(graph.vertices[u]),
          k,
        );
        if (dist > 2000) continue;
        mp[u] = sum(mp[u], force);
        mp[v] = sub(mp[v], force);
      }

      for (const j in graph.adjMatrix[u]) {
        const v = graph.adjMatrix[u][j];
        if (graph.vertices[v].idx > graph.vertices[u].idx) continue;
        const force =
↪  findAttractive(Vector.fromVertex(graph.vertices[v]),
↪  Vector.fromVertex(graph.vertices[u]), k);
        mp[u] = sub(mp[u], force);
        mp[v] = sum(mp[v], force);
      }
    }
    for (const u in graph.vertices) {
      const norm = mp[u].dist();
      if (norm < 1) continue;
      const tmp = Math.min(norm, t);
      const diff = mul(mp[u], tmp / norm);
      graph.vertices[u].x += diff.x;
      graph.vertices[u].y += diff.y;
    }
    t = t > 1.5 ? t * 0.85 : 1.5
  }
  return graph;
};
```

Листинг 5: Реализация функции нахождения отталкивающей силы

```
const findRepulsive = (
  lhs: Vector,
  rhs: Vector,
  k: number,
): [Vector, number] => {
  const delta = sub(rhs, lhs);
  const dist = delta.dist();
  const force = (k * k) / dist;
  return [mul(delta, force / dist), dist];
};
```

Листинг 6: Реализация функции нахождения притягивающей силы

```
const findRepulsive = (
  lhs: Vector,
  rhs: Vector,
  k: number,
): [Vector, number] => {
  const delta = sub(rhs, lhs);
  const dist = delta.dist();
  const force = (k * k) / dist;
  return [mul(delta, force / dist), dist];
};
```

Листинг 7: Реализация алгоритма Флойда-Уоршела

```
const floydWarshall = (graph: Graph): number[][] => {
  const verticesCount = graph.vertexCount;

  let dist = new Array(verticesCount);

  for (let i = 0; i < verticesCount; ++i) {
    dist[i] = new Array(verticesCount);

    for (let j = 0; j < verticesCount; j++) {
      dist[i][j] = Number.MAX_SAFE_INTEGER / 2 - 1;
    }
  }

  for (const u in graph.vertices) {
    const i = graph.vertices[u].idx;

    dist[i][i] = 0;

    for (const v in graph.adjMatrix[u]) {
      const vertex = graph.vertices[graph.adjMatrix[u][v]];
      dist[i][vertex.idx] = 1;
      dist[vertex.idx][i] = 1;
    }
  }

  for (let k = 0; k < Object.keys(graph.vertices).length; k++) {
    for (let i = 0; i < Object.keys(graph.vertices).length; i++) {
      for (let j = 0; j < Object.keys(graph.vertices).length; j++) {
        dist[i][j] = Math.min(dist[i][j], dist[i][k] + dist[k][j]);
      }
    }
  }

  return dist;
};
```

Листинг 8: Метод поиска значения энергии вершины

```
  private findVertexEnergy = (v: Vertex): number => {
    let xEnergy = 0.0;
    let yEnergy = 0.0;

    const vVec = Vector.fromVertex(v);
    for (const id in this.graph.vertices) {
      const u = this.graph.vertices[id];
      const uVec = Vector.fromVertex(u);
      if (u.idx == v.idx) continue;

      const delta = sub(vVec, uVec);
      const dist = delta.dist();

      const spring = this.springs[v.idx][u.idx];
      xEnergy += delta.x * spring.strength * (1.0 - spring.length /
↪   dist);
      yEnergy += delta.y * spring.strength * (1.0 - spring.length /
↪   dist);
    }

    return Math.sqrt(xEnergy * xEnergy + yEnergy * yEnergy);
  };
```

Листинг 9: Метод поиска вершины с максимальным значением энергии

```
  private findMaxVertexEnergy = (): [number, string] => {
    let max = -1.0;
    let maxId = "";
    for (let id in this.graph.vertices) {
      const energy = this.findVertexEnergy(this.graph.vertices[id]);
      if (energy > max) {
        maxId = id;
        max = energy;
      }
    }
    return [max, maxId];
  };
```

Листинг 10: Метод поиска нового положения вершины

```
private findNextVertexPos = (v: Vertex): Vector => {
    let xxEnergy = 0.0,
      xyEnergy = 0.0,
      yxEnergy = 0.0,
      yyEnergy = 0.0;
    let xEnergy = 0.0,
      yEnergy = 0.0;

    const vVec = Vector.fromVertex(v);
    for (const u in this.graph.vertices) {
      const uVertex = this.graph.vertices[u];
      const uVec = Vector.fromVertex(uVertex);
      if (uVertex.idx == v.idx) continue;

      const delta = sub(vVec, uVec);
      const dist = delta.dist();

      const spring = this.springs[v.idx][uVertex.idx];

      xEnergy += delta.x * spring.strength * (1.0 - spring.length /
↪   dist);
      yEnergy += delta.y * spring.strength * (1.0 - spring.length /
↪   dist);
      xyEnergy +=
        (spring.strength * spring.length * delta.x * delta.y) /
        Math.pow(dist, 3);
      xxEnergy +=
        spring.strength *
        (1.0 - (spring.length * delta.y * delta.y) / Math.pow(dist, 3));
      yyEnergy +=
        spring.strength *
        (1.0 - (spring.length * delta.x * delta.x) / Math.pow(dist, 3));
    }

    yxEnergy = xyEnergy;
    const denominator = xxEnergy * yyEnergy - xyEnergy * yxEnergy;
    vVec.x += (xyEnergy * yEnergy - yyEnergy * xEnergy) / denominator;
    vVec.y += (xyEnergy * xEnergy - xxEnergy * yEnergy) / denominator;

    return vVec;
  };
```

Листинг 11: Реализация алгоритма Камады-Кавай

```
public run = (width: number, height: number): Graph => {
    let steady_energy_count = 0;
    let [maxEnergy, id] = this.findMaxVertexEnergy();

    while (
      maxEnergy > this.eps &&
      steady_energy_count < MAX_STEADY_ENERGY_ITERS_COUNT
    ) {

      let vertex_count = 0;
      do {
        let newPos = this.findNextVertexPos(this.graph.vertices[id]);
        this.graph.vertices[id].x = newPos.x;
        this.graph.vertices[id].y = newPos.y;
        vertex_count++;
      } while (
        this.findVertexEnergy(this.graph.vertices[id]) > this.eps &&
        vertex_count < MAX_STEADY_ENERGY_ITERS_COUNT
      );

      let prevMax = maxEnergy;
      [maxEnergy, id] = this.findMaxVertexEnergy();
      if (Math.abs(maxEnergy - prevMax) < 1e-20) {
        steady_energy_count++;
      } else {
        steady_energy_count = 0;
      }
    }

    return centerAndScale(this.graph, width, height);
};
```

Листинг 12: Реализация функции draw

```
const draw = (graph: Graph, cy: cytoscape.Core) => {
  Object.keys(graph.vertices).forEach((id) => {
    const vertex = graph.vertices[id];
    cy.add({
      group: "nodes",
      data: { id: id, fullName: vertex.fullName, avatar: vertex.photoSrc
↪   },
      position: {
        x: graph.vertices[id].x,
        y: graph.vertices[id].y,
      },
    });
  });

  Object.keys(graph.adjMatrix).forEach((lhs) => {
    graph.adjMatrix[lhs].forEach((rhs) => {
      cy.add({
        group: "edges",
        data: { id: lhs + rhs, source: lhs, target: rhs },
      });
    });
  });
};
```