*Муниципальное общеобразовательное бюджетное учреждение*

*«Средняя общеобразовательная школа №3»*

Научно-практическая конференция учащихся

«Гагаринские чтения»

Исследование графов и выбор оптимального алгоритма для нахождения кратчайшего пути.

Выполнили ученики 10б класса

Шенягин Даниил и Субботин Андрей

Руководитель – учитель информатики

Краев Николай Владимирович

Муром – 2019

Содержание

[**Актуальность исследования:** 3](#_Toc506487161)

[**Цель работы:** 3](#_Toc506487162)

[**Задачи:** 3](#_Toc506487163)

[**Введение** 4](#_Toc506487164)

[**История изобретения графов** 5](#_Toc506487165)

[**Алгоритмы для графов** 6](#_Toc506487166)

[История разработки web-приложения 7](#_Toc506487167)

[Основная часть нашей работы 8](#_Toc506487168)

[Практическое применение нашей работы 11](#_Toc506487169)

[Вывод 12](#_Toc506487170)

# **Введение**

Когда мы готовились к ЕГЭ по информатике, столкнулись с задачами для решения которых, используются математические модели - графы. И у нас возникли вопросы: что такое графы, где они используются, и где их можно применять. Нам стало интересно более подробно изучить данный вопрос. А так же научиться применять их, графы, как и алгоритмы, на практике

Как оказалось, эта тема очень объемная и мы решили ее изучить.

Родоначальником теории графов принято считать математика Леонарда Эйлера (1707-1783), который решил задачу *о Кенигсбергских мостах.*

Напомню, задача состоит в том, чтобы обойти все четыре части суши, пройдя по каждому мосту один раз, и вернуться в исходную точку. Эта задача была решена (показано, что решение не существует) Эйлером в 1736 году.

Графы – абстрактная модель или конечная совокупность точек, именуемых вершинами; некоторые из них соединены друг с другом линиями, называемых ребрами графа. В математике существует целый раздел – теория графов, который изучает графы, их свойства и применение. Графы делятся на 2 основных типа: ориентированные и НЕориентированные. Мы же будем использовать неорграфы для простоты и экономии времени.

Каждый из нас сталкивался с графами. К примеру при взгляде на географическую карту сразу бросается в глаза сеть железных и автомобильных дорог. Это типичный граф: кружочки обозначают станции или города — это вершины графа, а соединяющие их пути - ребра.

Если вам, к примеру, нужно найти кратчайший путь из точки А в точку Б, может возникнуть проблема: чем больше промежуточных точек, тем больше вариантов маршрутов будет получаться и одному человеку просчитать кратчайший путь будет очень сложно и долго.

И поэтому мы решили изучить способы решения этой задачи.

Для решения этой задачи существуют алгоритмы, предназначенные для нахождения кратчайшего пути (далее - НКП). Но хочется сказать, что мы еще затронули алгоритмы для нахождения компоненты связанности, которые будут нужны в алгоритмах НКП и являющиеся второстепенными алгоритмами нашего исследования.

## **Цель работы:**

И мы поставили для себя цель:

Найти и выяснить какой алгоритм для нахождения кратчайшего пути в графе более оптимальный.

Критерии оптимальности:

1. Лучшее время работы алгоритма
2. Универсальность алгоритма
3. Простота работы
4. Практическое применение в навигационных картах

Далее мы составили задачи нашего исследования, которые вы видите на слайде.

# **Алгоритмы для графов**

Итак, мы выбрали для исследования 5 основных алгоритмов для НКП, которые вы можете видеть на слайде:

1. Алгоритм Дейкстры
2. Алгоритм Флойда — Уоршелла
3. Алгоритм Джонсона
4. Алгоритм Ли(волновой алгоритм)
5. Алгоритм Беллмана — Форда

Мы не стали брать алгоритм Ли или волновой алгоритм, т.к. он предназначен для нах. кр. пути в лабиринте. Итак, мы взяли для исследования 4 самых популярных алгоритмов для нахождения кратчайшего пути в графе:

1. Алгоритм Дейкстры
2. Алгоритм Флойда – Уоршелла
3. Алгоритм Форда – Беллмана
4. Алгоритм Джонсона

# **История разработки web-приложения**

Для нашего исследования мы спроектировали и создали web-frontend-приложение. Оно позволит быстро создавать и визуализировать графы, а также применять на них алгоритмы, также выводить статистику данного графа.

В двух словах: визуализация представлена на HTML5, ход дизайнерской идеи был реализован на CSS3 (Less или Sass), контроллером служила сеть компонентов, связанные между собой, работающая на React JS (Twitter).

Приложение визуально состоит из двух частей: рабочая область и панель управления и статистики.

Основными представляемыми функциями нашего приложения являются:

1. Визуальное создание графов
2. Вывод управляющей информации, расположенной в панели управления

Если останется время, я познакомлю вас с основным интерфейсом и расскажу о нашем приложении

# **Основная часть нашей работы**

Основная часть нашей работы разделяется на две подчасти:

* Теоретическая часть
* Практическая часть

Алгоритм исследования 1 части:

1. Вычисление теоретического времени
2. Генерация графов
3. Применение алгоритмов на графе
4. Обработка результатов
5. Вывод на основе результатов

Для начала давайте рассмотрим теоретическую часть.

Она заключается в генерации графов и испытание на них алгоритмов. Далее получение результатов, сравнение их, распределение и сортировка. Затем выявление оптимально алгоритма, который показывает наилучший результат по 4 характеристикам.

Одна из характеристик – это лучшее время работы, оно же в свою очередь подразделяется на два пункта: теор. время и прак. время. Но чтобы приступать к получению практического времени, для начала нужно рассчитать теоретическое время.

**Этап 1. - Расчет теоретического времени**  
Для каждого алгоритма существует теоретическое время работы, называемое временной сложностью алгоритма.

Зависит от архитектуры алгоритма, кол-ва входных данных и характеристик процессора, в результате мы получаем теоретическое время.

Для большей точности результатов мы решили не убирать константы из выражения big O.(O(V3) == O(V3 + E))

Временная сложность - это функция показывающая, как возрастает график от входных данных N(n). Функция будет возвращать какое-то число, и это число в теории будет количеством тактов процессора. И поэтому, узнав тактовую частоту процессора компьютера, можно легко высчитать теоретическое время работы любого алгоритма.

**Этап 2. Генерация графа**  
Для любого графа принимается 2 аргумента (кол-во вершин и кол-во ребер), притом, что этот граф будет не ориентированным и с положительным весом ребер.  
Мы создавали 13 графов, где начальное кол-во вершин – 10, а конечное – 70, с каждым разом кол-во вершин увеличивалось на 5. Почему именно мы остановились до 70, а не к примеру до 100. В данном примере мы использовали небольшое количество вершин и максимальное количество ребер для данного графа. А для большого количества ребер требуется большое место для памяти и более производительный процессор. Компьютер, на котором производилось исследование не справлялся с количеством вершин более 70, а может и справлялся, но уже подводил браузер. Для такого небольшого количества вершин мы брали максимальное количество ребер — а это такое количество ребер в графе, чтобы все вершины были соединены со всеми другими. Алгоритм для нахождения максимального количества ребер мы не нашли, и ,понаблюдав за увеличением вершин и ребер, мы составили собственный простейший алгоритм. Это задали рекурентной формулой:

* a0=3,
* i=3,
* an+1=an + i,
* i++(i с каждым разом увеличивается на единицу)

**Этап 3. Испытание графа на алгоритмы**

К сгенерированному графу 2 раза применяется каждый алгоритм. При этом происходит замер практического времени на выполнение. После чего генерируется новый граф.

Данные действия будут выполняться для каждой вершины(кол-во = N). При этом, параллельно этим действиям, будет считаться практическое время алгоритма для одной вершины и среднее время для всех вершин графа. Кол-во тестингов = кол-ву вершин(это рассмотрено в следующих этапах).

После этого мы получаем результаты и записываем их в таблицу.  
Проанализировав полученные результаты.

1. График теоретического времени отличается от графика практического времени от 100 до 10000 раз.
2. Алгоритмы которые по теоретическому времени работали быстрее, на практики работают медленее.
3. По практическому времени самый быстрый это алгоритм Флойда-Уоршела.

Проведя исследования, мы так и не поняли от чего такие большие отличия результатов, но выдвинули гипотезу, что такие различия могли образоваться от того, что:

1. Мы взяли не самый производительный по времени работы язык программирования - JS.
2. Мы использовали Web-приложение, которое должно исполнятся браузером, из-за которого все ровно теряется производительность.
3. Из вышесказанного про временную сложность алгоритмов: алгоритм исполнялся процессором (какое-то опр. Кол-во тактов), а процессор мог быть занят другими процессами, которые могли бы оказать некоторое влияние на производительность.

# Практическое применение нашей работы

Нам стало интересно испытать нашу работу на практическом применении. Посмотрев на карту метро, можно увидеть самый обыкновенный граф, где вершины графа будут являться станциями, а ребра путями. Суть проста – выбираем любую станцию из которой нужно добраться и выбираем другую станцию, в которую нужно добраться.

1 этап. Генерация графа

Взяв картинку метро, мы спроектировали граф.

2 этап. Использование алгоритмов.

Результаты остались те же, но алгоритм Флойда позволяет нам сохранить этот путь, что дает ему преимущество перед другими алгоритмами.

Скрины и таблицы приложены к практической части.

# Вывод

Проект получился весьма успешным и имеет колоссальный потенциал к развитию. Наша работа включает далеко не все алгоритмы, как для нахождения кратчайшего пути, так и для нахождения компоненты связанности.

Но хочется заметить, что после проведения нашей исследовательской работы у нас опять остались вопросы связанные с графами, а также с этой исследовательской работой. Например, почему же результаты теоретического и практического времена так различаются, причем несхождения очень велики. Также, почему по теоретическому времени самые быстрые алгоритмы будут самыми медленными по практическому?