Министерство образования и науки РФ

ФГБОУ ВО «Удмуртский государственный университет»

Институт математики, информационных технологий и физики

Кафедра алгебры и топологии

ВИЗУАЛИЗАЦИЯ ГЕОМЕТРИЧЕСКИХ АЛГОРИТМОВ

И АЛГОРИТМОВ НА СТРУКТУРАХ ТИПА ГРАФ

Выпускная квалификационная работа

(Магистерская диссертация)

студента магистерской программы

«Математические основы компьютерных наук»

Пучкова Ильи Александровича

Научный руководитель:

к. ф.-м. н., доцент Бастрыков Е.С.

Ижевск – 2017

Научный руководитель: к. ф.-м. н., доцент Бастрыков Е.С.

Рецензент: Технический директор компании «ЭЛМА» Кононов А.А.

Допускается к защите

Руководитель  
магистерской программы: д. ф.-м. н., профессор Грызлов А.А.

**Содержание**

стр.  
Введение 4

1. Графический редактор «Raphael» 6

2. Текстовый редактор кода «Ace» 7

3. О механизме визуализации 8

4. Разработка пользовательского API 8

4.1. Алгоритм построения выпуклой оболочки 9

4.2. Алгоритм поиска в глубину 11

4.3. Алгоритм поиска в ширину 12

4.4. Пример применения пользовательского API 13

5. Разработка красивого пользовательского интерфейса 16

6. Пример работы с инструментом 17

Заключение 24

Список использованных источников 25

Приложение (диск)

**Введение**

. С каждым годом компьютерные технологии проникают всё в большее количество сфер жизни человека. И сейчас мы уже не представляем себе жизни без компьютеров и программ на них работающих. Перед программистами и архитекторами программных продуктов ставятся всё новые цели: если в прошлом веке основной упор делался на оптимизации программ как по вычислительной сложности, так и по затратам памяти, в начале этого века камнем преткновения стали распределённые системы, трендом же последнего десятилетия является написание параллельных и конкурентных программ. Но как и в математике: для того чтобы изобрести что-то новое необходимо в совершенстве владеть методами, механиками и подходами уже придуманными, отработанными и доказавшими свою эффективность. Понимание базовых алгоритмов позволяет разрабатывать программы на совершенно другом уровне, позволяя программисту оперировать бо́льшими объектами системы. Именно поэтому изучение алгоритмов является важным шагом для любого человека, желающего стать специалистом в области написания хороших программ. С другой стороны рассмотрение алгоритмов всегда связывается со структурами данных. Это вытекает из тех ограничений, которые привнесли компьютеры. В математике мы можем просто сказать: «множество объектов». И человеку это будет понятно. А компьютеру нет. Также как фраза «упорядочим объекты множества по какому-то правилу» не требует дополнительных пояснений для человека, для компьютера же мы должны написать программу, с помощью которой он сможет эти объекты отсортировать. И выбор структуры данных определяет набор действий, которые с ней возможно производить и как следствие зависит от того что мы предполагаем с этими данными делать. И многие алгоритмы вытекают из выбранной структуры данных и наоборот. Таким образом, умение правильно выбрать структуру данных также важное умение для профессионала.

Одной из важнейших структур данных является граф. Человеческое мышление можно представить в виде графа, где вершинами являются объекты, а ребра — это ассоциации. В современном мире также получили развитие социальные сети, которые также представляют собой графы, где вершинами являются люди, вещи, интересы, медиа-контент, а ребрами — отношения. С другой стороны интернет представляет собой огромный граф, где ребра-соединения взвешены по множеству различных метрик, таких как скорость, пропускная способность, качество и так далее. И на этих структурах каждый день появляются всё новые задачи, многие из которых были решены 40–50 лет назад. Но у графов есть и свои недостатки: при разборе алгоритмов, работающих на графах, необходимо очень много информации держать в голове. Давайте сравним два алгоритма: простая сортировка вставками и обход графа в ширину. В первом случае необходимо держать в голове массив данных, указатель на неотсортированную часть, а также указатель на поиск места в отсортированной части. Один исходный массив и две дополнительные переменные. Во втором случае помимо графа необходимо помнить о массиве отметок, массиве родителей, а также об очереди, вынутой из неё вершине и о соседях это вершины. Хотя «на пальцах» оба алгоритма довольно просты, при пошаговом построении алгоритм на графе оказывается на порядок сложнее. И мне была поставлена задача разработать инструмент, который помогает в разборе алгоритмов на графах, а именно визуализирует работу с ними.

Для реализации проекта был выбран язык программирования javascript с использованием html. Javascript – относительно простой и быстро развивающийся язык программирования. С помощью этого языка программирования можно легко и удобно создавать графические интерфейсы, так как javascript поддерживает технологию canvas. Он запускается в браузере и не требует каких-либо дополнительных компиляторов или интерпретаторов.

1. **Графический редактор «Raphael»**

Работа над проектом велась в несколько этапов. В первую очередь, необходимо было выбрать инструмент для работы с векторной графикой. Для работы с векторной графикой есть несколько библиотек javascript. Среди кандидатов были такие библиотеки, как SVG.js, D3.js. и Raphael.js. Выбор был сделан в пользу Raphael.js, так как данная библиотека использует внутри себя SVG и VML, что позволяет ей работать на большом количестве браузеров (Internet Explorer 6.0+, Chrome 5.0+, Firefox 3.0+, Opera 9.5+, Safari 3.0+). По данной библиотеке есть хорошая документация. Хоть она и написана на английском языке, она довольно-таки проста для понимая.

Работа была начата с добавления поля графического редактора. Так как первоначально редактор предназначался для рисования геометрического места точек, то нужно было предоставить пользователю легко и быстро рисовать точки. Самым простым и удобным решением стало рисование точек по клику мыши. Точка рисовалась в том месте, где был курсор. В дальнейшем такой способ рисовать точки станет одним из возможных режимов рисования.

При использовании этого режима возникла первая проблема, связанная с удобством – постоянно рисовать объект на геометрическом месте точек было весьма накладно, так как занимало много времени. Для решения данной проблемы были добавлены функции выгрузки и загрузки входных данных. Данные сохраняются в формате JSON в файл с соответствующим расширением.

Следующим шагом нужно было добавить возможность рисовать графы. Для этого необходимо предоставить удобный способ рисования линий. Рисование линий происходит следующим образом: кликом выбирается первая точка (начало линии), затем кликом выбирается вторая точка (конец линии). Рисование линий было вынесено в отдельный режим, чтобы избежать конфликтов при рисовании точек и линий посредством кликов мыши. И было добавлено следующее ограничение: в режиме рисования точек нельзя рисовать линии и наоборот – в режиме рисования линий нельзя рисовать точки.

1. **Текстовый редактор кода «Ace»**

Предполагалось, что в инструмент будет встроено несколько алгоритмов. Однако это бы ограничивало возможности пользователя для визуализации других алгоритмов. Чтобы добавить инструменту гибкости, было принято решение добавить возможность пользователю самому писать интересующие его алгоритмы. Для комфортного написания алгоритмов необходимо встроить текстовый редактор кода, и лучшим кандидатом на эту роль стал текстовый редактор под названием «Ace».

Ace (сокр. от Ajax.org Clound9 Editor) – онлайн-редактор кода, написанный на javascript. Обладает подсветкой синтаксиса, настраиваемыми темами, горячими клавишами, автоматическим дополнением кода и многим другим. Его можно без проблем встроить в любую веб-страницу.

Данный редактор кода используется в большом количестве проектов, в том числе и на популярном портале github.com. Встраивание такого популярного редактора в мой проект существенно упрощает пользователям писать код, так как им не придётся переучиваться на новый редактор кода, а пользоваться уже знакомым инструментом. Так же была добавлена кнопка выполнения кода, написанного в редакторе.

Чтобы пользователю было понятно, с чем он имеет дело, были добавлены встроенные алгоритмы – построение выпуклой оболочки, поиск в ширину, поиск в глубину. Над редактором кода появился селектор, из которого можно выбрать интересующий вас алгоритм, и весь его исходный код появится в редакторе. Затем можно этот код выполнить, и запустить анимацию, чтобы посмотреть, как этот код работает.

1. **О механизме визуализации**

Мало просто показать визуально, как работает тот или иной алгоритм. Необходимо ещё каким-то образом предоставить текстовое описание происходящего шага. Именно для решения этой проблемы было принято решение добавить поле для отображения текстового описания действия. Добавление текстового описания происходит строго при выполнении какого-либо действия.

Одна из особенностей javascript – моментальное выполнение кода. Это и преимущество, и недостаток одновременно. Для пошаговой анимации алгоритма это явный недостаток – весь код выполняется мгновенно, и не успеваешь отследить, как именно происходит алгоритм. Именно из-за этой особенности было принято решение все основные моменты визуализации помещать в определённую очередь, элементы из которой потом пошагово доставать и визуализировать с помощью определённых функций. Вызов эти функций реализован с помощью кнопок.

Кнопка «Запустить анимацию» запускает цикл, который бежит по очереди событий. Особенность в том, что данная функция не визуализирует данный первый шаг, а лишь ставит указатель на первый элемент в данной очереди.

Кнопка «Перейти к следующему шагу» забирает элемент из очереди и визуализирует его.

Кнопка «Остановить анимацию» возвращает указатель в начало алгоритма, а также возвращает графический редактор к состоянию перед первым шагом анимации.

1. **Разработка пользовательского API**

Главным этапом разработки проекта было проектирование и написание пользовательского API, с помощью которого пользователь сам бы смог писать интересующие его алгоритмы. Но чтобы реализовать грамотный пользовательский API, сначала необходимо написать алгоритмы без его использования. Так как у нас уже есть готовые пользовательские алгоритмы, они и станут отправной точкой при формировании API. Для этого необходимо проанализировать написанные алгоритмы, найти части кода, которые повторяются наибольшее количество раз, и вынести их в отдельные функции. Кроме этого, в функции так же выносится логика, сложная для понимая пользователя. Опираясь на эти два фактора, встроенные алгоритмы были переписаны с использованием API. В результате код данных алгоритмов стал чище и понятнее, при этом не потерял своей функциональности.

**4.1. Алгоритм построения выпуклой оболочки**

Итак, рассмотрим встроенные геометрические алгоритмы. Начнём с построения выпуклой оболочки. Алгоритм построения выпуклой оболочки – это алгоритм на геометрическом месте точек. Введём несколько основных определений.

Определение. Геометрическим местом точек называется математическая фигура речи, которая употребляется для определения геометрической фигуры как множества точек, обладающих некоторым свойством.

Определение. Множество Х в векторном пространстве называется выпуклым, если оно содержит вместе с двумя любыми точками соединяющий их отрезок.

Определение. Выпуклой оболочкой множества точек Х называется минимальное выпуклое множество, содержащее Х.

Определение. Минимальной выпуклой оболочкой называется выпуклая оболочка наименьшего периметра.

Переходим к алгоритму построения выпуклой оболочки:

1. Помещаем все исходные точки в массив.
2. Выбираем из текущего множества точек стартовую точку – точку с наименьшими х и у координатами (обозначим как R).
3. Помещаем её номер в список номеров искомых точек.
4. Сортируем оставшиеся точки по степени их левизны относительно стартовой точки R:
   1. Берем точки с номерами i и j = i+1(обозначим A и B соответственно)
   2. Строим вектор RA, где начало вектора – наша стартовая точка R, а конец вектора – точка с номером i.
   3. Вычисляем векторное произведение для вектора RA и точки B
   4. Если векторное произведение меньше нуля, то точка B лежит левее точки A. Следовательно, меняем их местами
5. Записываем в список номеров искомых точек номер точки с индексом 1 из списка номеров отсортированных точек
6. Пробегаемся по оставшимся точкам в списке отсортированных точек:
   1. Строим вектор KL, где К – предпоследняя точка из списка искомых точек, а L – последняя точка из списка искомых точек.
   2. Вычисляем векторное произведение для вектора KL и точки O, где O – точка из исходного массива точек
   3. Пока векторное произведение будет меньше нуля, удаляем последнюю точку из списка искомых точек
   4. Как только векторное произведение стало больше нуля, добавляем текущую точку в список искомых точек
7. Получаем список искомых точек. Соединяя их по порядку, получаем минимальную выпуклую оболочку.

Сложность алгоритма составляет O(N\*logN), где N – количество точек.

Применяется в построении триангуляций; в распознавании образов; в обработке изображений; в задаче раскроя и компоновки материалов.

**Итог:** анализируя данный алгоритм, можно выделить несколько вещей, которые стоит добавить в пользовательское API для работы с геометрическим местом точек. К ним относятся – отметка точки (выделение определённым цветом), отметка линии (выделение определённым цветом), создание и удаление линий (которые соответствуют векторам в алгоритме).

**4.2. Алгоритм поиска в глубину**

Переходим к следующему алгоритму – поиск в глубину. Поиск в глубину является алгоритмом на структуре типа граф. Введём несколько основных определений.

Определение. Графом называется совокупность непустого множества вершин и связей между вершинами – рёбер.

Определение. Две вершины, соединённые одним ребром, называются смежными.

Переходим к алгоритму поиска в глубину:

1. Пройдём по всем вершинам графа. Пусть u – это вершина графа;
2. Если вершина u не была просмотрена, выполняем для неё функцию DepthFirstSearch(u);

Функция DepthFirstSearch(u):

1. Отмечаем u вершину как просмотренную;
2. Для любой вершины w, смежной с вершиной u, рекурсивно вызываем фунцию DepthFirstSearch(w).

Сложность алгоритма оставляет O(n+m), где n – количество вершин, а m – количество рёбер.

Применяется в качестве подпрограммы в алгоритмах поиска одно- и двусвязных компонент; в топологической сортировке; для поиска точек сочленения, мостов; в различных расчётах на графах.

**4.3. Алгоритм поиска в ширину**

Остался последний встроенный алгоритм – поиск в ширину. Это так же алгоритм на структуре типа граф. Описание алгоритма:

1. Пройдём по всем вершинам графа. Пусть u - это вершина графа;
2. Если вершина u не была просмотрена, добавляем её в очередь;
3. Пока очередь не пуста, выполняем функцию BreadthFirstSearch().

Функция BreadthFirstSearch():

1. Достаём вершину u из очереди;
2. Ищем вершины, смежные с вершиной u;
3. Если смежные вершины не найдены, помечаем вершину u просмотренной;
4. Иначе все смежные с вершиной u вершины, которые ещё не просмотрены и не находятся в очереди, добавляются в очередь. Вершина u помечается как просмотренная;
5. Возвращаемся к пункту 1.

Сложность алгоритма составляет O(n+m), где n – количество вершин, m – количество рёбер.

Применяется для поиска компонент связности в графе; для поиска кратчайшего пути между двумя узлами невзвешенного графа; является часть волнового алгоритма поиска пути в лабиринте; является частью алгоритма Форда-Фалкерсона (нахождение максимального потока).

**Итог:** анализируя данные алгоритмы, можно выделить несколько вещей, которые стоит добавить в пользовательское API для работы с графами. К ним относятся - отметка точки (выделение определённым цветом), отметка линии (выделение определённым цветом), получение всех соседей определённой точки, а также получение всех вершин графа.

**Общий итог:** проанализировав встроенные алгоритмы, получаем следующий набор требований к API:

1. Для работы с геометрическим местом точек нам понадобятся методы для подсветки точки, создания, удаления и подсветки линий, функцию вычисления векторного произведения;
2. Для работы с графами нам понадобятся методы подсветки точек и рёбер, метод получения всех соседей определённой точки, а также свойство для получения всех вершин графа.

**4.4. Пример применения пользовательского API**

Давайте напишем код для алгоритма поиска в ширину с использованием API.

API для алгоритмов графа называется Graph. Для его использования нужны выполнить следующий код:

var graph = new Graph();

Данный вызов проанализирует объект, нарисованный в графическом редакторе, а также создаст удобную модель для использования графа.

В первую очередь, нам нужно пробежаться по всем вершинам графа. Для этого в API есть свойство, позволяющее достать все вершины графа. Данное свойство называется vertices. Код для прохождения по всем вершинам графа выглядит следующим образом:

for (var i = 0; i < graph.vertices.length; i++) { … }

Далее идёт проверка вершины, которую достали из очереди. Для отметки данного действия при анимации, необходимо вызвать функцию markVertex, и последовательно передать туда вершину, которую необходимо отметить; цвет, которым необходимо пометить данную вершину; текст, который будет использован для описания шага. Выглядит это следующим образом:

graph.markVertex(vertex, selected, "Достаём вершину из очереди. Проверяем, были ли эта вершина просмотрена ранее.");

Если вершина не была просмотрена, для неё идет поиск смежных вершин. Отметим данное действие рассмотренной выше функцией.

graph.markVertex(vertex, selected, "Данная вершина не была просмотрена. Ищем соседей данной вершины.");

var neighbors = graph.getNeighbors(vertex);

Функция getNeighbors(vertex) достанет все вершины, смежные с переданной вершиной.

Далее просматриваются все смежные вершины, если таковые существуют. Если вершина не была просмотрена, её нужно добавить в очередь. Вершина, которая просматривается в данный момент, отмечается подсветкой соответствующего ребра. Такой подход сделан для удобства.

Для подсветки ребра необходимо использовать функцию markEdge и последовательно передать туда вершину – начало ребра; вершину – конец ребра; цвет, которым необходимо пометить данное ребро; текст, который будет использован для описания шага. Код выглядит следующим образом:

graph.markEdge(vertex, neighbor, selected, "Данная вершина не была просмотрена, добавляем её в очередь.");

В зависимости от того, была ли точка отмечена, передаваемый текст будет изменяться.

Все основные элементы API по работе с графами были рассмотрены. Используя их, написанный код для поиска в ширину имеет следующий вид:

var graph = new Graph();

var queue = new Array();

for (var i = 0; i < graph.vertices.length; i++) {

var startVertex = graph.vertices[i];

if (!startVertex.Viewed){

queue.push(startVertex);

}

while (queue.length > 0) {

var vertex = queue[0];

queue = queue.slice(1);

graph.markVertex(vertex, selected, "Достаём вершину из очереди. Проверяем, были ли эта вершина просмотрена ранее.");

if (!vertex.Viewed){

graph.markVertex(vertex, selected, "Данная вершина не была просмотрена. Ищем соседей данной вершины.");

var neighbors = graph.getNeighbors(vertex);

if (neighbors.length > 0){

graph.markVertex(vertex, selected, "Просматриваем соседей данной вершины.");

}

for (var j = 0; j < neighbors.length; j++){

var neighbor = neighbors[j];

if (!neighbor.Viewed){

if (!neighbor.InQueue){

neighbor.InQueue = true;

queue.push(neighbor);

graph.markEdge(vertex, neighbor, selected, "Данная вершина не была просмотрена, добавляем её в очередь.");

graph.markVertex(neighbor, inQueue, "Вершина добавлена в очередь.");

} else {

graph.markEdge(vertex, neighbor, selected, "Данную вершину пропускаем, так как она уже находится в очереди.");

}

} else{

graph.markEdge(vertex, neighbor, selected, "Данную вершину пропускаем, так как она была просмотрена.");

}

}

graph.markVertex(vertex, viewed, "Отмечаем точку как просмотренную.");

vertex.Viewed = true;

} else{

graph.markVertex(vertex, viewed, "Данную вершину пропускаем, так как она была отмечена ранее.");

}

}

}

**5. Разработка красивого пользовательского интерфейса**

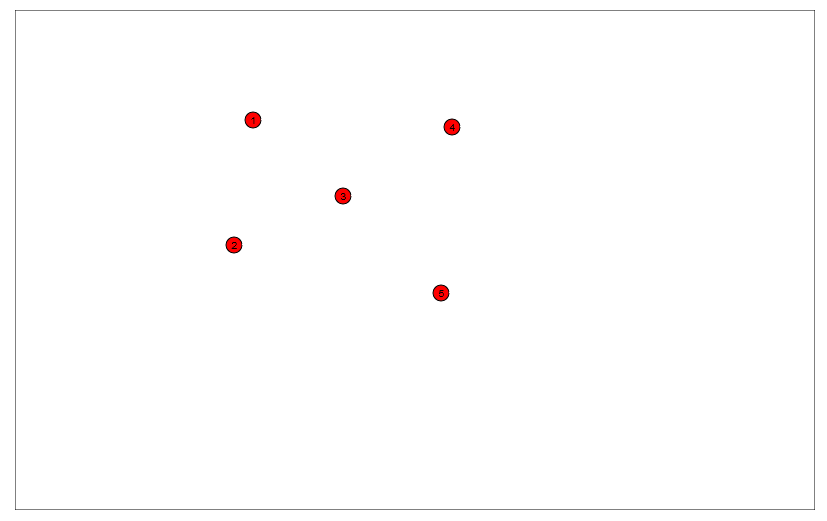
Последним шагом разработки было добавление красивого и понятного интерфейса. Задача «сделать инструмент красивым» была поставлена слишком поздно, но решать её нужно было быстро. У меня довольно мало опыта в веб-разработке и вёрстке html-страниц, а на изучение всех премудростей данной сферы просто не хватило бы времени. Для решения этой проблемы было принято использовать библиотеку Bootstrap (так же известный как Twitter Bootstrap), которая позволяет быстро и красиво писать код, поддерживаемый всеми актуальными браузерами.

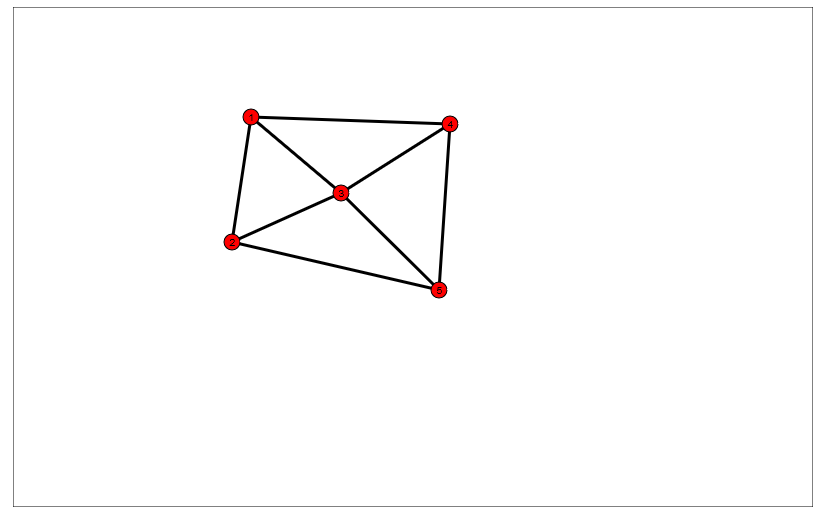
**Bootstrap** представляет собой несколько файлов с расширениями .css и .javascript, что позволяет с лёгкостью расширить возможности стандартного функционала. Самым главным плюсом данной библиотеки является простота разработки. Можно писать простой и понятный код, который будет понятен всем – от ученика старших классов с уклоном в информатику до опытного веб-разработчика.

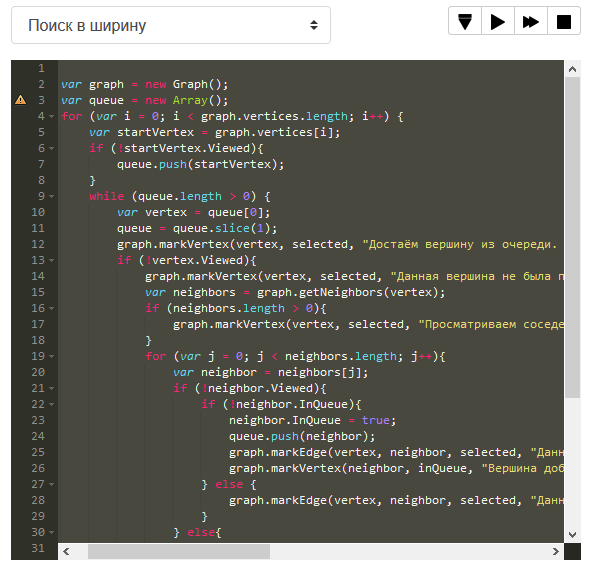
Используя готовые стили, скрипты и модели, реализация красивого интерфейса не заняла много времени, и не пришлось сильно углубляться в тонкости и прочие детали веб-разработки.

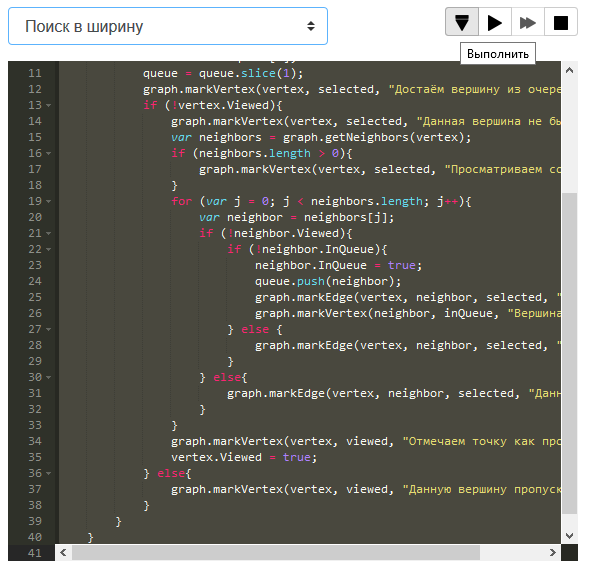
**6. Пример работы с инструментом**

Давайте разберём на примере, как работать с данным инструментом. Для примера рассмотрим анимацию поиска в ширину. В первую очередь, нужно нарисовать граф, на котором будет выполняться поиск в ширину. Рисуем вершины нашего графа. Для этого необходимо поставить курсор мыши в нужное место и кликнуть.

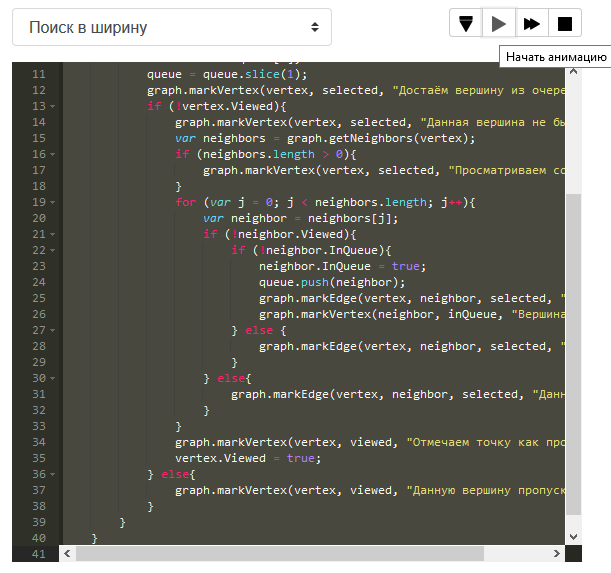


Затем переключаем режим рисования и рисуем рёбра графа. Для этого нужно выделить две вершины – начальную и конечную.

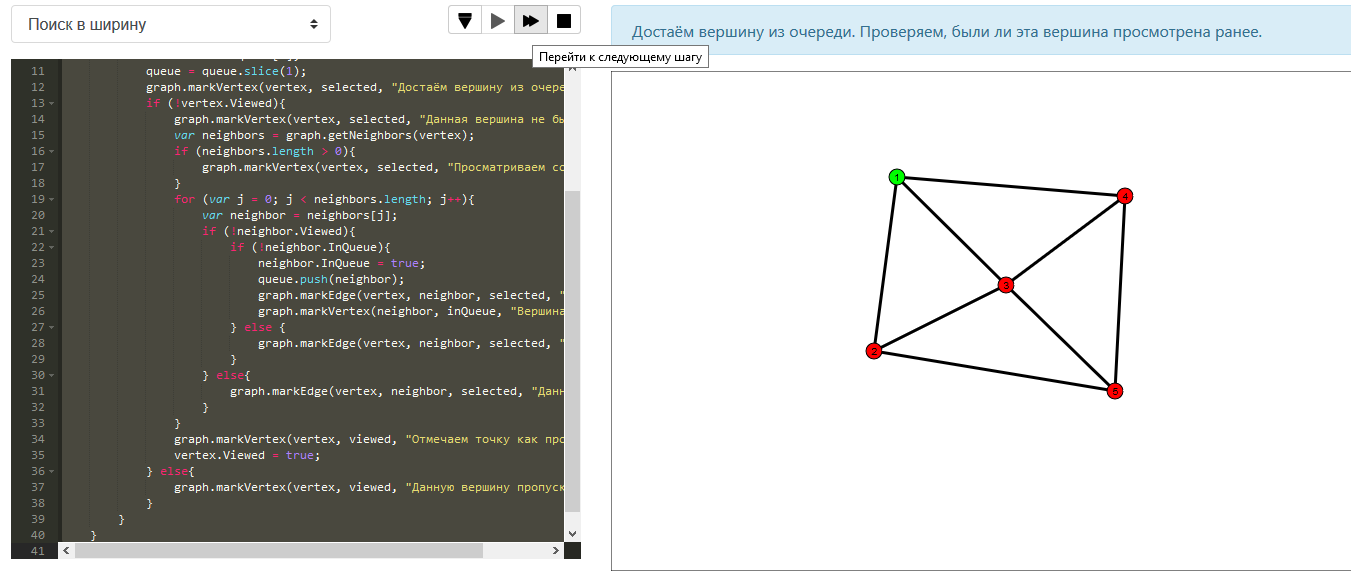
Выбираем один из встроенных алгоритмов.

Нажмите на кнопку «Выполнить», это действие выполнит код, написанный в редакторе, и подготовит очередь для визуализации алгоритма 

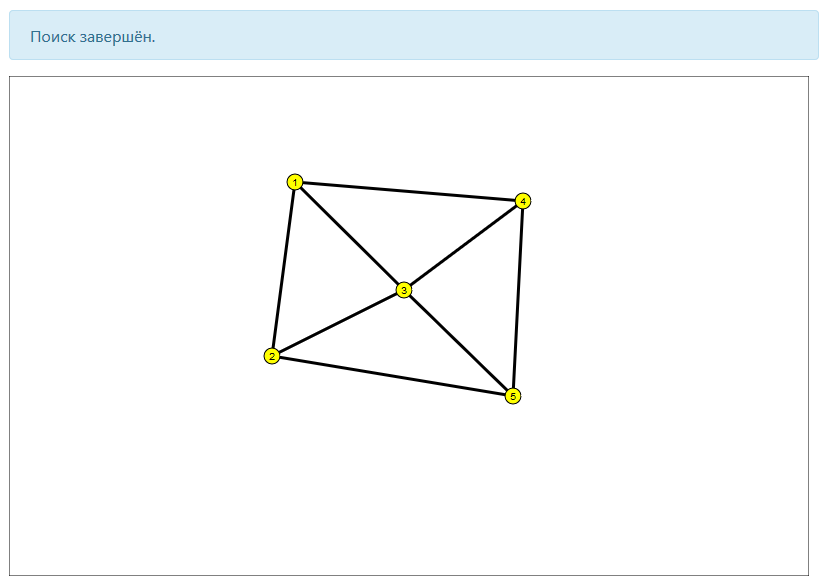
Нажмите на кнопку «Начать анимацию», чтобы начать визуализацию алгоритма. Данное действие подготовит инструмент к пошаговой визуализации алгоритма.



Чтобы перейти к просмотру шага алгоритма, нажмите кнопку «Перейти к следующему шагу». Данное действие выполнит первый шаг алгоритма, произведя описанные действия с нарисованным объектом и запишет текстовое описание шага в соответствующее поле.



Если же вы в какой-то момент сбились с алгоритма, можно без проблем вернутся к его началу. Нажмите на кнопку «Остановить анимацию». Это действие вернёт графические редактор к состоянию перед первым шагом анимации. После нажатия данной кнопки алгоритм начнёт визуализацию с самого начала.

Когда алгоритм закончится, в поле отображения текстовой информации вы увидите соответствующее сообщение.

Более подробное описание работы с инструментом можно найти в справке.

**Заключение**

Нами разработан инструмент для визуализации алгоритмов на графах и ГМТ. Он выполняет поставленные задачи: позволяет, используя API визуализировать, любой алгоритм введённый пользователем, а также в графическом режиме редактировать входные данные. Также для удобства тестирования инструмента были добавлены функции выгрузки и загрузки входных данных, что не входило в изначально поставленные цели, однако оказалось очень удобным. Также механизм визуализации был реализован не так как планировалось изначально, то есть не «вживую» во время выполнения алгоритма, а с помощью записи событий, что позволяет реализовать новый функционал: движение не только вперёд по шагам алгоритма, но и назад. В целом, проект представляет из себя оформленный результат. Оформленный, но не законченный, потому как по мере реализации изначальных целей появлялись новые, достичь которых в рамках двух лет магистратуры мы просто не успели. Вот некоторые возможности, которые хотелось бы добавить в проект: движение по истории как вперёд, так и назад; более простое рисование графов; возможность работать со взвешенными и ориентированными графами; вывод дополнительных структур данных (стеки и очереди, используем в алгоритмах).

Исходный код инструмента можно скачать тут: <https://github.com/SteamUpdate/geometricapi.github.io>

**Список использованных источников**

1. Асанов М.О. Дискретная математика: графы, матроиды, алгоритмы / М.О. Асанов, В.А. Баранский, В.В. Расин – Ижевск: НИЦ «РХД», 2001. – 288 с.
2. Скиена С.С. Алгоритмы. Руководство по разработке / С.С. Скиена – 2-е изд.: Перю с англ. – СПб.: БХВ-Петербург, 2013. – 720 с.
3. Ершов Н. Построение минимальных выпуклых оболочек [Электронный ресурс] // <http://habrahabr.ru/> [Интернет-портал]. 02.06.2012 г. URL: <http://habrahabr.ru/post/144921/> (дата обращения: 16.04.2014)