МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02. Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Системное программирование и компьютерные технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Исследование алгоритмов решения задачи определения изоморфизма графов»

Студент П.Н. Орлова
(И.О. Фамилия) (личная подпись)

Руководитель М.А. Тренина
(ученая степень, звание, И.О. Фамилия)

Консультант К.А. Селиверстова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Аннотация

Выпускная квалификационная работа посвящена исследованию инвариантов графов с целью выявления наиболее полного при определении изоморфизма.

Ключевые слова: граф, инвариант графа, изоморфизм.

Задача проверки двух графов на предмет изоморфизма является одной из ключевых задач в таких сферах деятельности и исследований человека, как компьютерная химия, проектирование электронных схем, логистика, и многие другие направления, использующие графы в качестве инструмента при выполнении поставленных задач. С ростом количества таких сфер исследований возрастает актуальность решения данного вопроса.

Вся задача определения изоморфизма графов сводится к двум методам: перебору всех возможных перестановок вершин графов, при совпадении которых у двух графов изоморфизм считается выявленным, и нахождении наиболее полного инварианта графа, из совпадения значений которого следовал бы изоморфизм.

Объект исследования – изоморфизм графов.

Предмет исследования – инварианты графа.

Целью выпускной квалификационной работы является исследование различных инвариантов на предмет выявления наиболее полного при определении изоморфизма графов, путём анализа результатов программы, автоматизирующей процесс генерации графов и вычисления их инвариантов.

В ходе выполнения выпускной работы было проведено исследование с применением разработанного программного приложения, и сделаны выводы с определением наиболее полного инварианта в условиях, описанных в рамках данной работы.

Выпускная квалификационная работа представлена на 58 страницах, включает 22 иллюстрации, 3 таблицы, 29 формул, список используемой литературы, состоящий из 21 источника и 1 приложение.

Abstract

The title of the graduation work is «Research of algorithms for solving the graph isomorphism recognition problem».

The graduation work consists of an explanatory note on 58 pages, includes 22 illustrations, 3 tables, 29 formulas, the list of 21 references including 6 foreign sources and 1 appendix.

The object of this work is isomorphism of graphs.

The subject of this work is invariants of graph.

The aim of the graduation work is to research various invariants in order to identify the most complete one when determining graph isomorphism by analyzing the results of a program that automates the process of generating graphs and calculating their invariants.

The graduation work may be divided into several logically connected parts. In the first part we consider all known methods for determining graph isomorphism, including iteration over all possible permutations of the vertices of the graphs and finding the most complete invariant of a graph. In the second part we describe the process of developing a software application, which we will use in future experiments. And finally, in the third part we present the results of experiments and on their basis we conclude which of invariants is the most complete.

As a result of the study, it was found that the vector of eigenvalues of adjacency matrix is considered the most complete invariant for graphs with characteristics described in the framework of this graduation work.

Содержание

Введение

В настоящее время необходимость проверки графов на изоморфизм возникает в различных областях исследований в таких направлениях как:

– компьютерная химия при решении химических задач с применением представления молекул в виде графов;

– автоматизация проектирования электронных схем при верификации различных их представлений в процессе проектирования микросхем;

– оптимизация программ при выявлении общих подвыражений;

– транспортная логистика при проектировании транспортных маршрутов;

– распознавание образов изображений при построении эталонного графа на основании информации о классе, к которому было отнесено исследуемое изображение, с которым будет производиться сравнение последующих создаваемых графов.

Выше представлены только некоторые области исследований, где вопрос распознавания изоморфизма графов возникает наиболее часто, но существуют также и другие сферы, использующие при решении своих задач графы, а, следовательно, в них так же может возникнуть необходимость распознавания изоморфизма графов. В связи с этим возрастает актуальность решения данной задачи.

В настоящее время существует два подхода к решению данной задачи – задачи определения изоморфизма графов [6]:

– перебор всех возможных комбинаций перестановок вершин графов с целью выявления совпадающей для обоих графов, что будет свидетельствовать об изоморфизме графов;

– отыскание полного инварианта графа, из совпадения значения которого однозначно бы следовало совпадение графов, то есть их изоморфность.

Объектом исследования в данной выпускной работе является задача определения изоморфизма графов.

Предметом исследования – инварианты графа.

Поскольку полного инварианта, вычислимого за полиномиальное время, на сегодняшний день не существует [6], целью данной выпускной квалификационной работы является исследование известных инвариантов графа [14] с целью выявления наиболее полного для решения задачи определения изоморфизма графов заданного типа.

Для достижения поставленной цели в рамках данной выпускной квалификационной работы необходимо решить следующие задачи:

1. Рассмотреть существующие алгоритмы решения задачи определения изоморфизма графов;
2. Рассмотреть существующие инварианты графов;
3. Реализовать один из известных алгоритмов перебора, выполняющий проверку графов на изоморфизм на языке программирования Java;
4. Реализовать вычисление инвариантов графа на языке программирования Java;
5. Провести эксперименты и анализ полученных результатов;
6. Выявить наиболее полный инвариант.

Данная выпускная квалификационная работа состоит из введения, трёх глав и заключения.

В первой главе представлено описание задачи определения изоморфизма графов, обзор существующих алгоритмов перебора и инвариантов графа.

Во второй главе описана программная реализация генерации графов, вычисления его инвариантов и одного из выбранных алгоритмов перебора.

В третьей главе представлены результаты экспериментов и их сравнительный анализ.

1 Обзор существующих методов определения изоморфизма графов

1.1 Общие понятия изоморфизма графов и постановка задачи на исследование

В теории графов изоморфизм графов определяется как биекция множества вершин графа G_1 на множество вершин графа G_2 с сохранением отношений смежности. Иными словами, граф G_1 изоморфен графу G_2 в том случае, когда существует такое взаимно однозначное соответствие между вершинами данных графов, при котором для любых двух смежных вершин первого графа найдутся соответствующие им вершины во втором графе, и они так же будут смежными между собой [7, 14]. На рисунке 1 представлены примеры изоморфных графов.

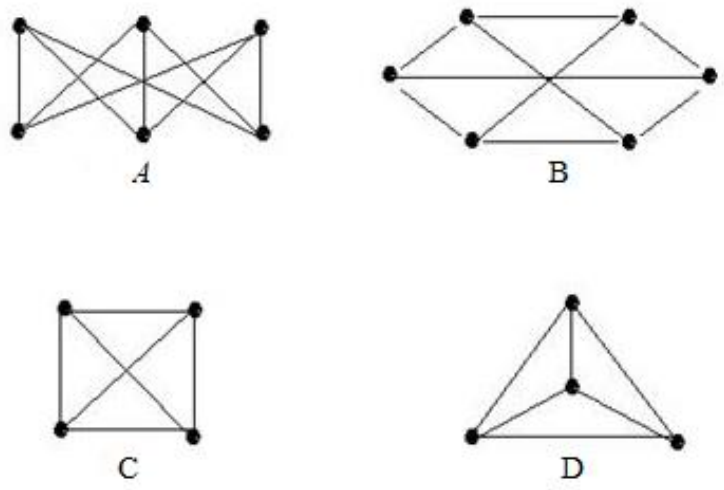


Рисунок 1 - Примеры изоморфных графов

Так, на данном рисунке изображены две пары изоморфных графов: граф *A* изоморфен с графом *B, C* изоморфен с *D.*

В настоящее время существует два подхода к решению задачи определения изоморфизма графов.

Первый подход основан на использовании теоремы: «Графы изоморфны тогда и только тогда, когда их матрицы смежностей можно получить одну из другой одинаковыми перестановками строк и столбцов» [6].

Он состоит в том, что вместо поиска эквивалентных перестановок вершин графов полным перебором всех возможных комбинаций их нумераций производится перестановка строк и столбцов в матрице смежности. Но в данном случае возникает проблема с вычислительной сложностью, так как при отсутствии изоморфизма необходимо произвести количество перестановок, равное факториалу количества вершин проверяемого графа. При достаточно большом количестве вершин это приведёт к очень большим временным затратам работы алгоритма полного перебора. Существуют алгоритмы, реализующие дополнительные проверки и накладывающие дополнительные условия на процесс определения изоморфизма, тем самым сокращая время вычислений. К данным алгоритмам относятся алгоритм Ульмана, алгоритм VF2, алгоритм полиномиального времени.

Второй подход основан на вычислении инварианта графа – некоторого числового значения или упорядоченного набора значений, позволяющего охарактеризовать структуру графа вне зависимости от его графического изображения или способа обозначения вершин [6]. Найти такой инвариант, из совпадения которого следовал бы изоморфизм графов – цель этого подхода. К недостаткам метода относится то, что на сегодняшний день не существует такого полного инварианта, который можно было вычислить за полиномиальное время.

В исследовании, производимом в рамках данной работы, будут использоваться неориентированные графы без петель и кратных рёбер с различным числом вершин. Также в данной работе будет реализован один из алгоритмов группы первого метода решения задачи определения изоморфизма, и будут исследованы инварианты на предмет отыскания

наиболее полного при определении изоморфизма графов с указанными выше характеристиками.

1.2 Обзор основных алгоритмов перебора, решающих задачу определения изоморфизма графов

1.2.1 Алгоритм Ульмана

В 1976 году Дж. Ульманом был разработан алгоритм, реализующий прямой перебор при решении задачи поиска изоморфных подграфов в некотором графе [19]. Впоследствии данный алгоритм был доработан и представлен в качестве решения задачи поиска изоморфизма [20].

Суть данного алгоритма [19] заключается в том, что производится перебор всех возможных соответствий вершин двух графов с реализацией поиска в глубину. Отличием этого алгоритма от обычного перебора всех возможных комбинаций является то, что в данном случае вводятся дополнительные условия и ограничения, позволяющие как можно раньше завершить вычисления и сравнения для не удовлетворяющего этим условиям набора. Также при использовании данного алгоритма подразумевается предварительная обработка входных данных – вектор степеней вершин исследуемого графа должен быть упорядочен в порядке убывания значений степеней вершин. Решение по данному алгоритму находится за экспоненциальное время [8].

Блок-схема алгоритма Ульмана представлена на рисунке 2.

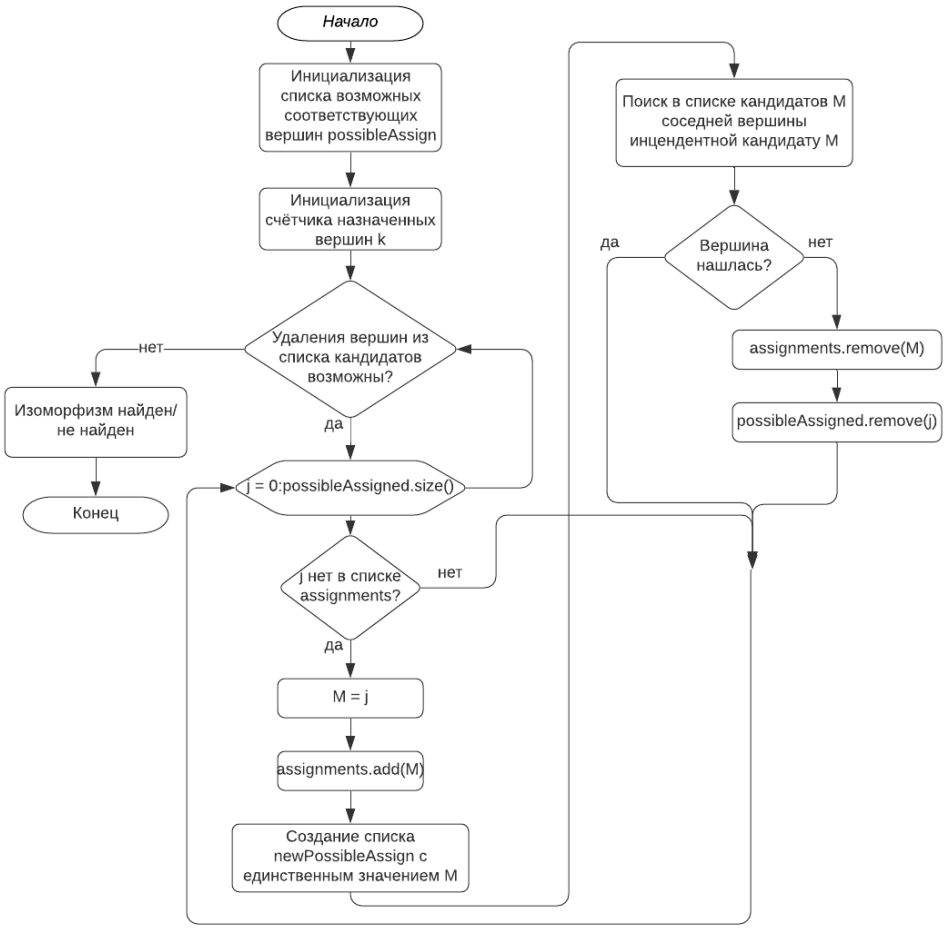


Рисунок 2 - Блок-схема алгоритма Ульмана

Исходя из выше изложенной блок-схемы алгоритма, можно заметить, что на первом этапе его выполнения составляются списки возможных соответствующих вершин графа G_2 для вершин графа G_1. Данные кандидаты выбираются исходя из необходимого свойства равенства степеней соответствующих вершин графов. Далее вступает в работу основная идея алгоритма Ульмана. Она заключается в том, что если некоторая вершина L из графа G_2 находится в списке возможных соответствий для некоторой вершины M из графа G_1, то у каждой соседней вершины M в списке кандидатов на соответствие должна быть хотя бы одна вершина, соседняя с L. При отсутствии выполнения данного условия вершина L удаляется из

списка возможных соответствий для вершины M. Данная проверка производится до тех пор, пока не станут невозможными удаления.

Таким образом, по данному алгоритму происходит отсечение неудовлетворяющих вариантов на ранних стадиях вычислений. Временная сложность алгоритма Ульмана для поиска изоморфного подграфа:

$$O(n!\,n^3), \tag{1}$$

где n – количество вершин исследуемого на изоморфизм графа.

Поскольку данный алгоритм [19] нацелен в большей степени на решение задачи поиска изоморфных подграфов и предоставляет решение за экспоненциальное время [8], было принято решение отказаться от реализации данного алгоритма.

1.2.2 Алгоритм VF2

Алгоритм VF2 является доработанной версией алгоритма Ульмана, осуществляющего определение изоморфизма графов и подграфов больших размерностей [16].

Результатом работы алгоритма является отображение $M(s)$, включающее в себя пары связанных вершин графа G_1 и G_2. В процессе нахождения функции отображения происходит оперирование понятием состояния. Каждое состояние s процесса сопоставления может быть связано с решением частичного отображения $M(s)$, которое содержит в себе не все связанные вершины графов, а только подмножество M.

Данный алгоритм вводит набор правил, предназначенных для проверки условия согласованности, делая возможным создание только согласованных состояний. Также число состояний может быть дополнительно уменьшено в процессе работы алгоритма путём применения другого набора правил – правил предварительного просмотра, производящих предварительную проверку на наличие у согласованного состояния s последующих согласованных преемников после k шагов выполнения алгоритма [16].

На рисунке 3 представлена блок-схема алгоритма VF2.

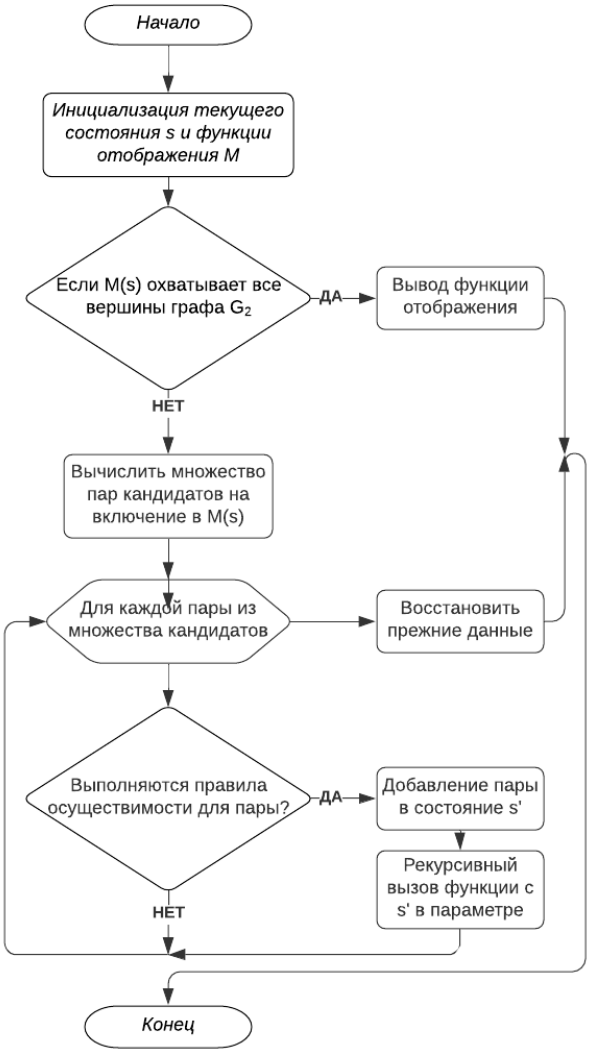


Рисунок 3 - Блок-схема алгоритма VF2

В начальном состоянии s_0 искомая функция отображения не содержит в себе никаких элементов. Для каждого промежуточного состояния *s* происходит вычисление списка пар узлов, которые будут являться кандидатами на добавление в текущее состояние *s*. Далее для каждой пары узлов оценивается выполнение правил осуществимости. Если данная пара удовлетворяет правилам, то она добавляется в состояние *s*. Далее весь процесс рекурсивно повторяется для последующих состояний и

производится построение дерева поиска. Также в алгоритме предусмотрена специальная процедура генерации преемника некоторого узла, позволяющая избежать генерирования бесполезных или уже сгенерированных состояний, что сокращает само формируемое дерево поиска.

Всего в алгоритме применяется 5 правил, два из которых используются при проверке непротиворечивости частичного решения $M(s')$, полученного при добавлении рассматриваемой пары кандидатов к текущему решению. Три других правила введены с целью обрезки дерева поиска, выполняющие предварительный просмотр.

Временная сложность алгоритма Ульмана для поиска изоморфного подграфа:

$$O(n!\,n), \tag{2}$$

где n – количество вершин исследуемого на изоморфизм графа.

Хотя данный алгоритм считается оптимизированной версией алгоритма Ульмана, предназначенной для работы с графами больших размерностей и со значительно сокращённым временем работы, в данной работе он так же не будет реализован, поскольку было найдено ещё более оптимальное решение задачи определения изоморфизма графов.

1.2.3 Алгоритм полиномиального времени

Алгоритм полиномиального времени решает задачу определения изоморфизма двух графов за число шагов, всегда ограниченное полиномиальной функцией, зависящей от размера входных данных, а именно, в данном случае, от количества вершин в графах [17].

В данном алгоритме помимо понятий вектора степеней вершин графа, матрицы смежностей и пары соответствующих вершин используются такие понятия, как множество рёбер E, матрица знаков и каноническая форма матрицы знаков.

Матрица знаков S некоторого графа G записывается как массив размера $n \times n$ с (u, v)-символами s_{uv} в качестве записи в строке u и столбце v и представляется в виде

$$S = [\, s_{uv} \,]. \tag{3}$$

Каноническая форма S^* знаковой матрицы – знаковая матрица S с упорядоченными определённым образом строками и столбцами. Сначала записывается множество всех различных (u, v)-символов s_{uv} в лексикографическом порядке $s_1, s_2, \ldots, s_r$. Затем для каждой строки $i = 1, 2, \ldots, n$ знаковой матрицы рассчитывается вектор частот знака f_i по формуле

$$f_i = \left(f_i^{(1)}, f_i^{(2)}, \ldots, f_i^{(r)}\right), \tag{4}$$

где $f_i^{(k)}$ – количество раз, когда знак s_k встречается в строке i.

Поскольку матрица S симметрична, то вектор частот знака для столбца i будет равен вектору частот знака строки i при $i = 1, 2, \ldots, n$. Далее записываются векторы частот знака $f_1, f_2, \ldots, f_n$ в лексикографическом порядке $f_{i_1}, f_{i_2}, \ldots, f_{i_n}$. И после изменения порядка строк и столбцов знаковой матрицы в соответствии с перестановкой $i_1, i_2, \ldots, i_n$ вершин $1, 2, \ldots, n$ графа G получается каноническая форма S^* знаковой матрицы.

По данному алгоритму вершины графа G разбиты на классы эквивалентности, состоящие из вершин с одинаковыми знаковыми векторами частот. Таким образом, каноническая форма S^* знаковой матрицы однозначно определяется только до перестановок вершин в каждом классе эквивалентности.

По данному алгоритму [17] граф G_A считается изоморфным графу G_B, если существует биекция φ от вершин V_A графа G_A к вершинам V_B графа G_B

$$\varphi: V_A \to V_B, \tag{5}$$

так что uv является ребром в графе G_A тогда и только тогда, когда $\varphi(u)$ $\varphi(v)$ является ребром в графе G_B.

Таким образом, по данному алгоритму, если графы G_A и G_B изоморфны, то они должны иметь одинаковые векторы частот знака в лексикографическом порядке $f_{i_1}, f_{i_2}, \dots, f_{i_n}$.

Теперь непосредственно опишем сам алгоритм. Его можно разделить на четыре отдельные процедуры, выполняющие некоторый набор действий и непосредственно сам алгоритм.

В первой процедуре применяется алгоритм Дейкстры [13]: вычисляются кратчайшие пути от вершины u до всех вершин v графа G, определяется значение $a(u, v)$, равное единице, если ребро uv принадлежит множеству рёбер E, и бесконечности в противном случае, заполняется множество вершин V_{known}, для которых известен кратчайший (u, v)-путь и множество предварительных расстояний $d'(u, w)$ для каждой вершины w, находящейся вне множества V_{known}. Блок-схема первой процедуры представлена на рисунке 4.

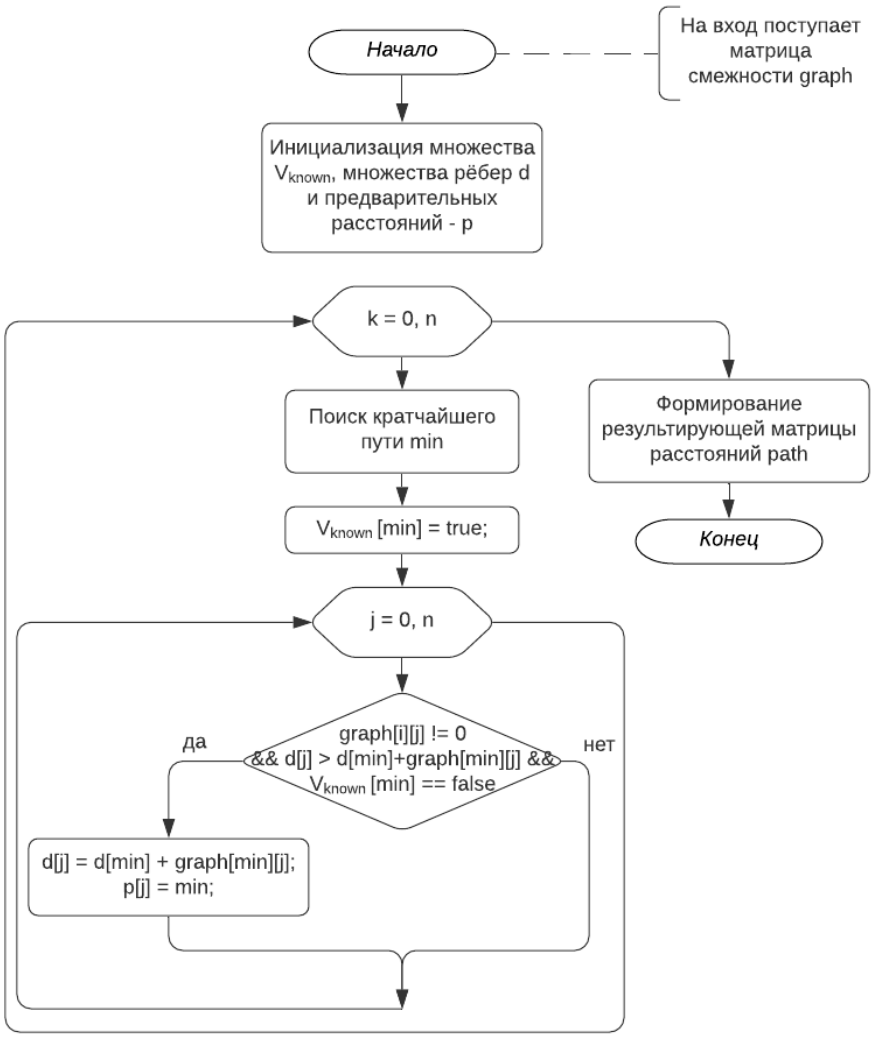


Рисунок 4 - Блок-схема первой процедуры алгоритма полиномиального времени

Временная сложность первой процедуры не превышает:

$$O(3n^2 + 3n), \tag{6}$$

где n – количество вершин исследуемого на изоморфизм графа.

Во второй процедуре производится вычисление расстояния $d(u, v)$ в побочном графе $G\backslash uv$ и парном графе Guv. Используя первую процедуру, вычисляются кратчайшие (u, x)-пути ко всем вершинам x графа $G\backslash uv$ и кратчайшие (v, y)-пути ко всем вершинам y графа $G\backslash uv$. Если $u = u_1, u_2, \dots, u_r$

и $v = v_1, v_2, \ldots, v_s$ – кратчайшие пути, такие что $u_r = v_s$, и сумма длин двух путей является расстоянием $d(u, v)$ в побочном графе $G\backslash uv$, тогда объединение вершин двух путей является вершинами парного графа Guv. Каждая вершина w пары графа Guv получается таким образом, что любой кратчайший (u, v)-путь, содержащий w, получается путём соединения некоторого кратчайшего (u, w)-пути с некоторым кратчайшим (w, v)-путём в графе $G\backslash uv$. Блок-схема второй процедуры представлена на рисунке 5.

Рисунок 5 - Блок-схема второй процедуры алгоритма полиномиального времени

Таким образом, хотя бы одна пара кратчайших путей, найденных выше описанным способом, должна удовлетворять равенству $u_r = v_s = w$ для каждой вершины графа *Guv*. Временная сложность второй процедуры не превышает:

$$O(7n^2 + 7n), \tag{7}$$

где n – количество вершин исследуемого на изоморфизм графа.

Третья процедура подразумевает вычисление матрицы знаков S и её канонической формы S^*. Используя вторую процедуру, для каждой пары вершин u и v вычисляется расстояние $d(u, v)$ в побочном графе $G\backslash uv$ и парном графе Guv. Запись s_{uv} в строке u и столбце v матрицы знаков S выглядит как

$$s_{uv} = \pm\, d_{uv} \cdot n_{uv} \cdot m_{uv}, \tag{8}$$

где старший двоичный знак положительный, если uv принадлежит множеству рёбер графа, или отрицательный в противном случае;

d_{uv} – расстояние $d(u, v)$ в побочном графе $G\backslash uv$;

n_{uv} – число вершин в парном графе Guv;

m_{uv} – число рёбер в парном графе Guv.

Далее записывается множество всех различных знаков s_{uv} в лексикографическом порядке $s_1, s_2, \ldots, s_r$. Для каждой строки $i = 1, 2, \ldots, n$ знаковой матрицы S вычисляется вектор частот знака по формуле (4), записывается в лексикографическом порядке $f_{i_1}, f_{i_2}, \ldots, f_{i_n}$ вектор частот знака $f_1, f_2, \ldots, f_n$, и изменяется порядок строк и столбцов знаковой матрицы в соответствии с перестановкой $i_1, i_2, \ldots, i_n$ вершин $1, 2, \ldots, n$ графа G. Блок-схема третьей процедуры представлена на рисунке 6.

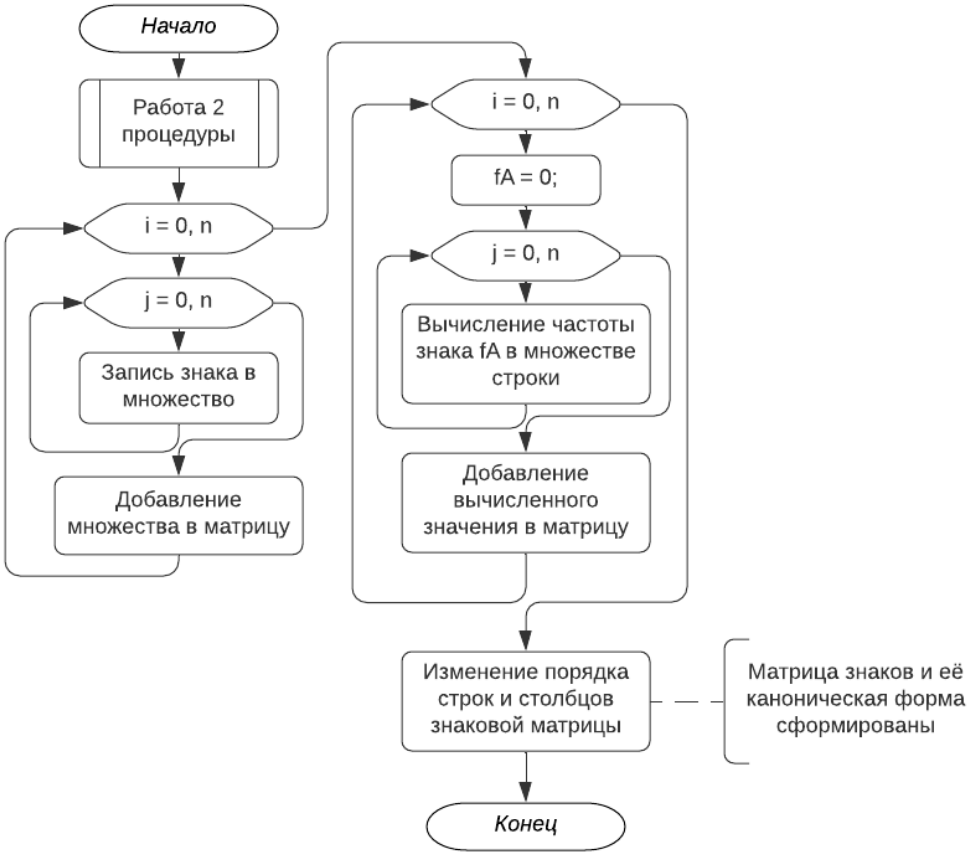


Рисунок 6 - Блок-схема третьей процедуры алгоритма полиномиального времени

Таким образом, в результате работы процедуры на выходе получается каноническая форма S^*. Временная сложность третьей процедуры не превышает:

$$O(7n^4 + 7n^3 + 2n^2), \quad (9)$$

где n – количество вершин исследуемого на изоморфизм графа.

В четвёртой процедуре устанавливается, что

$$A = S_{A^*},\ B = S_{B^*}, \quad (10)$$

где A – первая текущая рабочая матрица,

S_{A*} – каноническая форма матрицы A,

B – вторая текущая рабочая матрица,

S_{B*} – каноническая форма матрицы B.

Матрицы A и B считываются слева направо и сверху вниз. Если все соответствующие записи A_{ij} и B_{ij} матриц A и B совпали, то вычисления останавливаются. Иначе производится поиск первой записи B_{ij} в матрице B, не совпадающей с соответствующей записью A_{ij} в матрице A. Находится $k > i$ таким образом, чтобы перестановка строк (k, j) и столбцов (k, j) в матрице B гарантировала то, что первое несовпадение происходит после B_{ij}. Если такого k не существует, то вычисления останавливаются. Процесс повторяется до тех пор, пока соответствующее значение k не будет найдено или пока не совпадут записи матриц A и B. Блок-схема четвёртой процедуры представлена на рисунке 7.

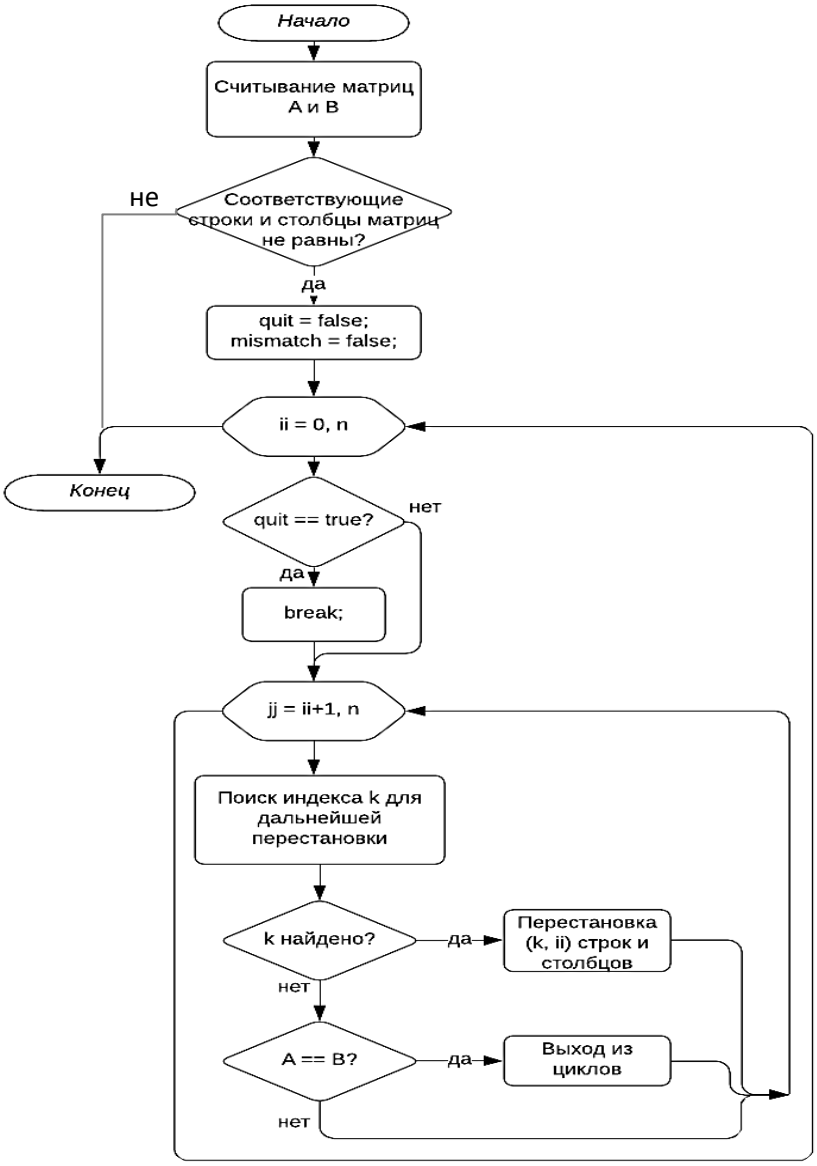


Рисунок 7 - Блок-схема четвёртой процедуры алгоритма полиномиального времени

Таким образом, в результате работы четвёртой процедуры на выходе получается переупорядочение $i''_1, i''_2, ..., i''_n$ вершин графа B так, что либо первая запись из B, которая не совпадает со соответствующей записью из A появляется с максимально возможным индексом в строке, либо устанавливается, что $A = B$. Временная сложность четвёртой процедуры не превышает:

$$O(2n^4), \quad (11)$$

где n – количество вершин исследуемого на изоморфизм графа.

Непосредственно сам алгоритм предназначен для формирования явной функции изоморфизма в случае его определения для данных графов G_A и G_B. В процессе его работы, используя третью процедуру, вычисляются канонические формы знаковых матриц S_{A*} и S_{B*}. В случае различия векторов частот знаков, упорядоченных в лексикографическом порядке, следует отсутствие изоморфизма графов G_A и G_B, и выполнение дальнейших действий прерывается. В случае совпадения данных векторов запускается цикл со счётчиком $k = 1, 2, \ldots, n$, в теле которого считываются матрицы A и B согласно формуле (10), меняются местами строки $(1, k)$ и столбцы $(1, k)$ матрицы B, запускается в работу четвёртая процедура и, если $A = B$ выполнение действий останавливается. Иначе начинаются вычисления со следующим значением k. При достижении $k = n$ вычисления останавливаются. Блок-схема алгоритма представлена на рисунке 8.

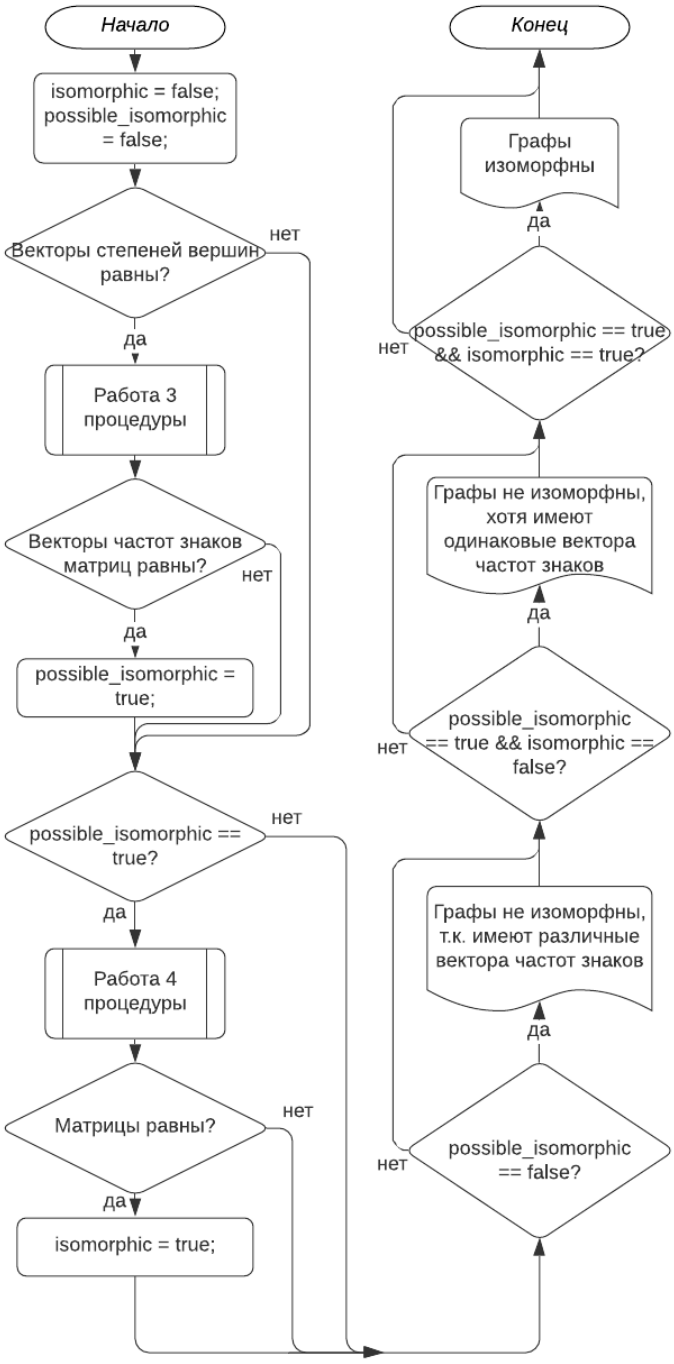


Рисунок 8 - Блок-схема определения изоморфизма графов по алгоритму полиномиального времени

Если в результате работы алгоритма было установлено, что $A \neq B$, то графы G_A и G_B не изоморфны. В противном случае, $A = B$, граф G_A изоморфен G_B и переупорядочение $i''_1, i''_2, ..., i''_n$ вершин графа G_B, предназначенное для

получения $S_{B*} = B$ обеспечивает явную функцию изоморфизма графов $\varphi(i_1) = i''_1$, $\varphi(i_2) = i''_2$, … , $\varphi(i_n) = i''_n$. Временная сложность алгоритма не превышает:

$$O(2n^5 + 14n^4 + 14n^3 + 4n^2), \quad (12)$$

где n – количество вершин исследуемого на изоморфизм графа.

Таким образом, рассмотренный алгоритм [17] позволяет решить задачу определения изоморфизма графов за полиномиальное время [8], что является значительным преимуществом в сравнении с производительностью других рассмотренных в данном разделе алгоритмов. Было принято решение реализовать данный алгоритм на языке программирования Java в рамках данной работы с целью точного определения изоморфизма графов для дальнейших исследований их инвариантов.

1.3 Обзор основных инвариантов графа

Инвариантом графа принято называть функцию f, относящую графу G некоторое значение $f(G)$ из множества произвольной природы, совпадающее для изоморфных графов, иными словами, для некоторых изоморфных графов G и G' будет справедливо равенство $f(G) = f(G')$ [7, 14].

В настоящее время известно более 20 инвариантов графа, но в данной работе будут исследованы только следующие:

- упорядоченный вектор степеней вершин,
- индекс Винера,
- индекс Рандича,
- диаметр графа,
- число компонент связности,
- хроматическое число,
- определитель матрицы смежности,

– упорядоченный вектор собственных чисел матрицы смежности.

1.3.1 Вектор степеней вершин графа

Степенью вершины u неориентированного графа G называется число $d(u)$ рёбер, инцидентных данной вершине u. Вектор степеней вершин графа содержит в себе значения степеней всех вершин графа. Сумма степеней некоторого графа $G=(V, E)$ равна удвоенной сумме рёбер:

$$\sum_{u \in V} deg(u) = 2|E|, \tag{13}$$

где V – множество вершин графа G,

E – множество рёбер графа G.

Из формулы (13) также следует свойство чётности количества вершин нечётной степени.

Последовательностью степеней вершин неориентированного графа принято считать невозрастающую последовательность. Графической последовательностью считается последовательность степеней вершин графа, реализацией которой является любой граф, имеющий идентичную степенную последовательность [18]. Графическую последовательность всегда можно считать правильной. Чтобы проверить, является ли заданная последовательность графической, можно использовать критерий Гавела-Хакими [4].

Пусть d – некоторая правильная последовательность с $n > 1$. Зафиксировав индекс i такой, что $1 \leq i \leq n$, получим последовательность c^i путём вычёркиванием из d i-ого члена, и получим производную последовательность d^i, уменьшив на 1 первые d_i члены в c^i. Согласно рассматриваемому критерию, если для последовательности d найдётся такое i, при которой производная последовательность d^i будет графической, то и последовательность d будет графической. Если последовательность d – графическая, то и любая последовательность d^i будет являться графической [4, 18].

Используя описанный выше критерий, в данной работе будет производиться проверка корректности вектора степеней $d = (d_1, d_2, \ldots, d_n)$, поступающего на вход алгоритму генерации графа [21]. Непосредственно генерация графа будет производиться на основе алгоритма генерации его матрицы смежности, блок-схема которого изображена на рисунке 9.

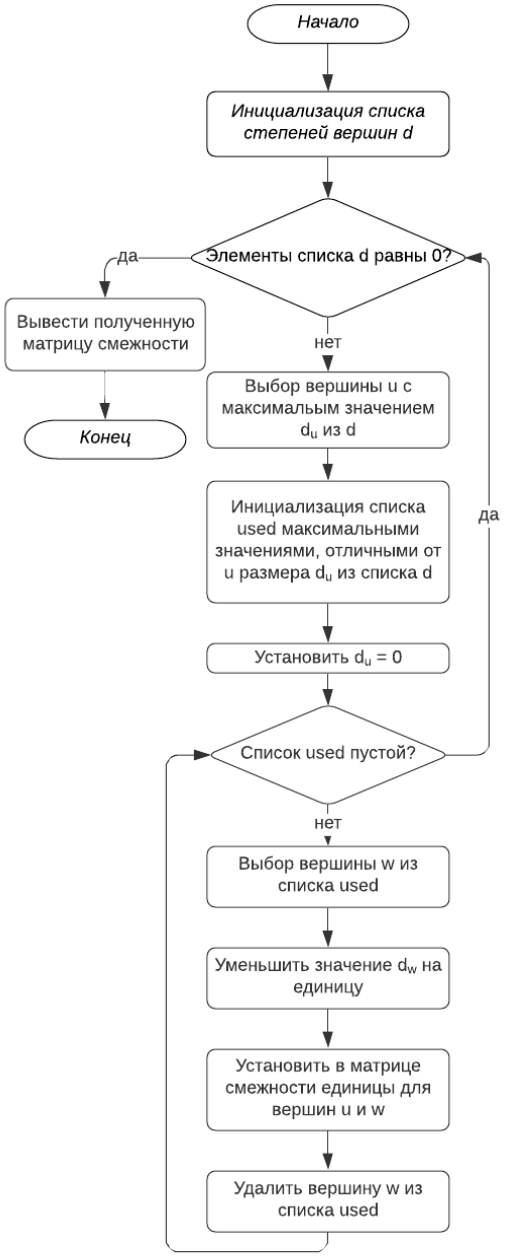


Рисунок 9 - Блок-схема алгоритма генерации матрицы смежности

На вход данному алгоритму будет поступать вектор степеней вершин графа $d = (d_1, d_2, \dots, d_n)$. Сам алгоритм, исходя из блок-схемы, можно описать следующей последовательностью шагов:

1. Выбирается вершина $u \in V$ такая, что $d_u > 0$ и максимально в векторе.

2. Выбирается d_u различных вершин, отличных от u.

3. Из списка выбранных вершин каждая w-ая вершина соединяется ребром с u-ой вершиной. Для соответствующих вершин устанавливаются единицы в матрице смежности.

4. Значение d_u во входном векторе d заменяется на значение 0.

5. В списке выбранных на втором шаге вершин уменьшаются значения d_w на единицу.

6. Шаги 1 – 5 выполняются до тех пор, пока элементы списка d не станут равными 0.

На выходе получается матрица смежности, по которой затем строится сам граф.

1.3.2 Индекс Винера

Индекс Винера $W(G)$ – инвариант неориентированного графа $G = (V, E)$, вычисляемый как сумма длин кратчайших расстояний $d(v_i, v_j)$ вершин v_i и v_j графа по формуле:

$$W(G) = \sum_{\forall i,j} d(v_i, v_j). \quad (14)$$

При построении матрицы достижимости для дальнейшего вычисления кратчайшего расстояния между вершинами графа [3] в данной работе использовался алгоритм Флойда-Уоршелла, блок-схема которого изображена на рисунке 10.

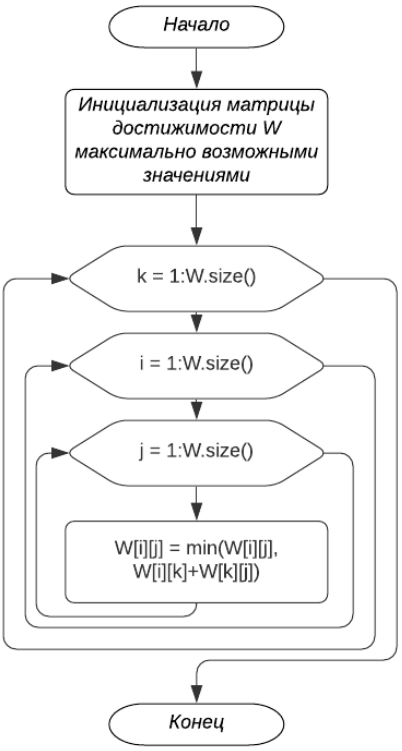


Рисунок 10 - Алгоритм формирования матрицы кратчайших путей по алгоритму Флойда-Уоршелла

Суть данного алгоритма заключается в проверке всех расстояний между вершинами графа u и v с целью выявления кратчайшего. Существует только два варианта, при которых может быть определено данное значение d_{uv}^{k}, $k = 1, 2, \dots, n$ – номера вершин графа:

– Если кратчайший путь является ребром между u и v и не проходит через дополнительную вершину w, тогда $d_{uv}^{k} = d_{uv}^{k-1}$;

– Если кратчайший путь проходит через вершину w, образуя тем самым путь от u до w, и затем от w до v и тогда расстояние, тогда $d_{uv}^{k} = d_{uw}^{k-1} + d_{wv}^{k-1}$.

Таким образом, по алгоритму Флойда-Уоршелла вычисляются все значения d_{uv}^{k} для $k = 1, \dots,$ n путём выбора минимального значения из двух возможных, описанных выше. Время выполнения алгоритма

$$O(n^3), \tag{15}$$

где n – количество вершин в исследуемом графе.

1.3.3 Индекс Рандича

Индекс Рандича $r(G)$ – инвариант неориентированного графа G, представляющий собой сумму вкладов по рёбрам, вычисляемую по формуле:

$$r(G) = \sum_{(v_i,v_j)\in V} \frac{1}{\sqrt{d(v_i)d(v_j)}}, \quad (16)$$

где v_i, v_j – вершины некоторого ребра,

$d(v_k)$ – степень вершины v_k.

Известен также под названием «индекс связности». Временная сложность вычисления

$$O(n), \quad (17)$$

где n – количество рёбер в исследуемом графе.

1.3.4 Диаметр графа

Диаметром графа G $diam(G)$ называют длину расстояния между двумя наиболее удалёнными друг от друга вершинами. При вычислении данного расстояниями было принято решение воспользоваться алгоритмом Флойда-Уоршелла, описанным в пункте 1.3.2, а именно сформированной по алгоритму, изображённому на рисунке 10, матрице кратчайших расстояний. Время выполнения алгоритма

$$O(n^3), \quad (18)$$

где n – количество вершин в исследуемом графе.

1.3.5 Число компонент связности

Число компонент связности графа G $k(G)$ – инвариант графа, содержащий в себе значение максимального связного, то есть содержащего единственную компоненту связности, подграфа графа G [7]. Иными словами, это количество подграфов графа G, содержащих его вершины, для любой пары которых в G имеется (u, v)-цепь, и не существует (u, w)-цепи для пары вершин, в которой w не принадлежит множеству вершин графа G.

Для определения этого числа в данной работе использовался алгоритм обхода графа в глубину [3, 9, 13], согласно которому происходит рекурсивный перебор всех рёбер, выходящих из рассматриваемой вершины, и если оно соединено с вершиной, не рассмотренной ранее, то запускается аналогичная процедура из этой вершины. Возвращение к изначальной вершине на каждой итерации рекурсии происходит в том случае, если следующая рассматриваемая вершина – лист, то есть не имеет дальше рёбер. При наличии незадействованных вершин, алгоритм запускается заново, начиная с такой вершины.

Временная сложность вычисления инварианта

$$O(n), \tag{19}$$

где n –сумма вершин и рёбер рассматриваемого графа.

1.3.6 Хроматическое число

Хроматическое число графа G $\chi(G)$ – инвариант графа, содержащий в себе значение минимального количества цветов, при применении которого граф будет иметь правильную раскраску вершин, то есть вершины одного ребра будут иметь разные цвета [10].

Для вычисления этой компоненты в данной работе была использована теорема Зыкова [7]: «Пусть имеется граф $G(V, E)$, в котором берутся две несмежные вершины u и v. Тогда хроматическая функция $h(G, t)$ будет являться суммой двух других хроматических функций $h(G_1, t)$ и $h(G_2, t)$, где

граф G_1 получается из графа G путём соединения этих двух несмежных вершин u и v новым ребром, а G_2 – их стягиванием».

Согласно этой теореме можно в явном виде описать хроматическую функцию графа.

На рисунке 11 представлена блок-схема алгоритма нахождения хроматического числа.

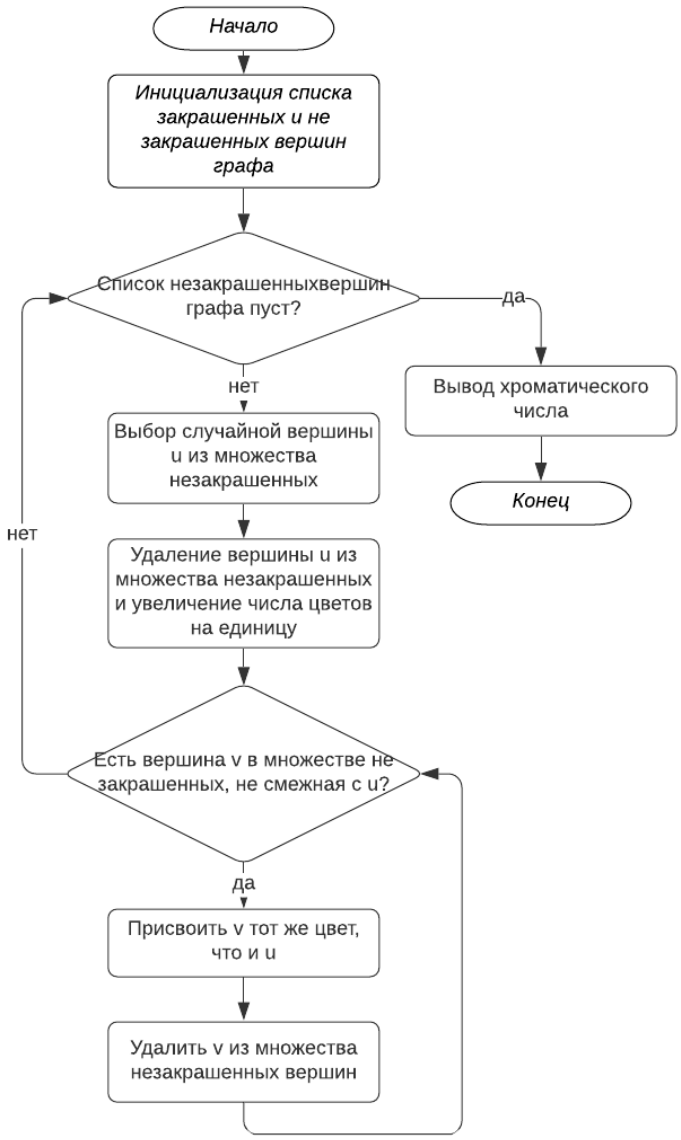


Рисунок 11 - Алгоритм нахождения хроматического числа, основанный на теореме Зыкова

Исходя из выше сказанного, алгоритм заключается в следующих шагах:

1. Выбирается случайная вершина u из множества не закрашенных вершин V_{uncol};

2. Выбранной вершине u присваивается новый цвет и увеличивается число использованных цветов на одну единицу;

3. Производится проверка не закрашенных вершин v, $v \in V_{uncol}$ на предмет смежности с закрашенной на предыдущем шаге u;

4. В случае отсутствия смежности вершине v присваивается тот же цвет, что и u;

5. Шаги 1 – 4 выполняются до тех пор, пока не будут закрашены все вершины исходного графа.

Временная сложность алгоритма

$$O(n), \tag{20}$$

где n – число вершин рассматриваемого графа.

1.3.7 Определитель матрицы смежности

Одним из инвариантов графа также принято считать определитель $det(A)$ матрицы смежности A графа G [7], поскольку он может быть поставлен в соответствие квадратной матрице, представляющей матрицу смежности. В настоящее время существует множество методов нахождения определителя квадратных матриц [5], но в данной работе было принято решение реализовать наиболее известный и эффективный метод – метод Гаусса [1, 12], блок-схема которого представлена на рисунке 12.

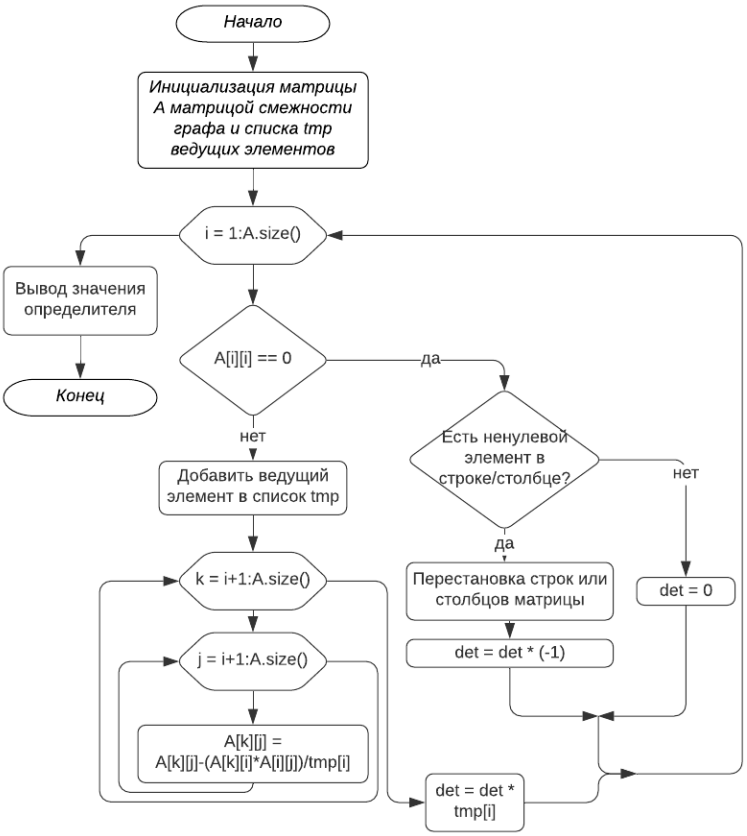


Рисунок 12 - Блок-схема нахождения определителя матрицы методом Гаусса

Алгоритм состоит из четырёх шагов [12]:

1. Выбирается ведущий элемент, стоящий на главной диагонали, не равный 0, элемент a_{11}. Если на главной диагонали находится элемент, равный 0, то производится перестановка текущей строки или столбца со строкой или столбцом с ненулевым элементом и знак определителя меняется на противоположный. Если такой строки или столбца нет – определитель равен 0, и вычисления останавливаются;

2. Ведущий элемент a_{11} записывается в список и выносится из первого столбца в качестве общего множителя определителя. Из элементов a_{ij} каждого j-ого столбца вычитаются соответствующие элементы a_{i1} первого столбца, умноженные на a_{1j}:

$$a_{ij}^{(1)} = a_{ij} - a_{i1} \cdot \frac{a_{1j}}{a_{11}}; \quad (21)$$

3. В преобразованном определителе элемент a_{11} будет равен 1, остальные элементы первой строки – нули, поэтому происходит переход к новому определителю без первой строки, и к нему применяются действия из шага 1;

4. Шаги 1 - 3 выполняются до тех пор, пока не будет пройдена вся матрица, после этого вычисляется сам определитель по формуле:

$$det\ A = a_{11} \cdot a_{22}^{(1)} \cdot \ldots \cdot a_{nn}^{(n-1)}, \quad (22)$$

где *det A* – непосредственно сам определитель,

$a_{ii}^{(k)}$ – ведущие элементы определителя порядка *k* из каждого шага выполнения алгоритма.

Временная сложность алгоритма не превышает значения

$$O(n^3), \quad (23)$$

где *n* – порядок матрицы смежности графа.

Поскольку у графов, используемых при исследовании и описанных в подразделе 1.1, отсутствуют петли – это свидетельствует об отсутствии элементов, отличных от нуля на главной диагонали матриц смежности графов. Из данного факта можно сделать преждевременный вывод о том, что в данном случае велика вероятность получения определителя матрицы смежности данного типа, равным 0, что повлечёт за собой дальнейшую нецелесообразность использования данного инварианта. Достоверность данного вывода будет проверена в дальнейших экспериментах.

1.3.8 Упорядоченный вектор собственных чисел матрицы смежности

Ещё одним инвариантов графа G является упорядоченный по возрастанию или убыванию вектор собственных чисел матрицы смежности. Некоторое число λ называется собственным числом, или собственным значением, матрицы A, если существует некоторый вектор $x = (x_1, x_2, \dots, x_n)$, удовлетворяющий равенству $Ax = \lambda x$ и называемый собственным вектором матрицы A, отвечающим собственным значениям λ. Поскольку матрица смежности любого рассматриваемого в данном исследовании графа является симметричной квадратной матрицей, то длина вектора собственных значений будет совпадать с порядком данной матрицы, а так как она симметричная, то все её собственные значения будут вещественными числами [11].

Для нахождения собственных чисел матриц существует две группы методов – итерационные и точные [1, 5, 11]. Поскольку в описываемом исследовании важна точность получения результатов вычислений для проведения дальнейших сравнений, и итерационные методы не в состоянии предоставить абсолютную точность вычислений, в данной работе было отдано предпочтение точному методу нахождения собственных чисел матрицы, а именно методу Данилевского, блок-схема которого изображена на рисунке 13.

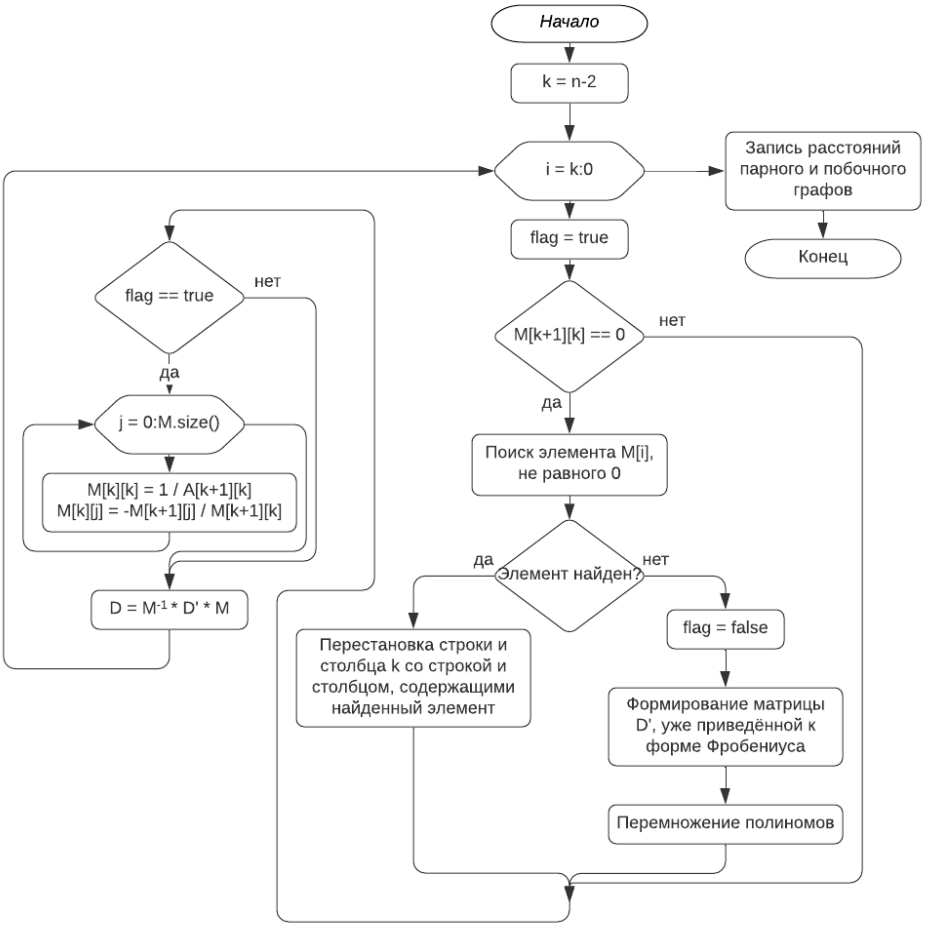


Рисунок 13 - Блок-схема алгоритма метода Данилевского для нахождения собственных чисел матрицы

Сущность данного метода заключается в преобразовании матрицы смежности в матрицу Фробениуса P [5] вида

$$P = \begin{pmatrix} p_1 & p_2 & \dots & p_{n-1} & p_n \\ 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}. \quad (24)$$

Это преобразование производится по формуле

$$P = B^{-1} \cdot A \cdot B, \quad (25)$$

где P – матрица формы Фробениуса;

B^{-1} – обратная матрица подобия,

A – матрица смежности графа,

B – матрица подобия.

Первым шагом алгоритма является проверка элемента $a_{n,n-1}$ матрицы A на отличие от 0. Если условие удовлетворяется, то начинается построение подобной матрицы B путём замены элементов n-1 строки в единичной матрице порядка n значениями, вычисленными по формулам

$$b_{n-1,j} = -\frac{a_{n,j}}{a_{n,n-1}}, \quad (26)$$

$$b_{n-1,n-1} = \frac{1}{a_{n,n-1}}, \quad (27)$$

где $b_{n-1,k}$ – элементы матрицы подобия B, стоящие на n-1 строке,

$a_{n,k}$ – элементы матрицы смежности графа, стоящие на n строке.

Дальнейшие вычисления матрицы подобия D_1 сводятся к перемножению матриц по формуле

$$D_1 = B_1^{-1} A B_1. \quad (28)$$

На втором шаге выполнения алгоритма вычисления производятся аналогичные, но со строкой, содержащей элемент $d_{n-1,n-2}$, и так для всех строк подобных матриц с каждой итерации выполнения алгоритма.

Если же условие отличия от 0 элемента из шага 1 не выполняется, то это является исключительной ситуацией метода Данилевского, и тогда проверяется отличие от нуля элемента $d_{k,m}$, где $m < k - 1$. Если условие выполняется, то происходит одновременная перестановка (k – 1)-ой и m-ой строки и (k – 1)-ого и m-ого столбцов, и продолжается применение метода Данилевского уже для новой матрицы подобия. Если условие не выполняется, тогда результирующая матрица D имеет следующий вид

$$D = \left(\begin{array}{cccc|cccc} & (D') & & & & & (F) & \\ d_{11} & d_{12} & \dots & d_{1,k-1} & d_{1k} & \dots & d_{1,n-1} & d_{1n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ d_{k-1,1} & d_{k-1,2} & \dots & d_{k-1,k-1} & d_{k-1,k} & \dots & d_{k-1,n-1} & d_{k-1,n} \\ \hline 0 & 0 & \dots & 0 & d_{kk} & \dots & d_{k,n-1} & d_{kn} \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & 0 & \dots & 1 & 0 \\ & (0) & & & & & (D'') & \end{array}\right) = \left(\begin{array}{c|c} D' & F \\ \hline 0 & D'' \end{array}\right). \quad (29)$$

В этом случае матрица D'' является уже приведённой к форме Фробениуса и необходимо продолжать выполнение метода Данилевского для матрицы D'.

Первая строка полученной матрицы Фробениуса P определяет коэффициенты характеристического уравнения матрицы A, из которого потом находятся собственные числа данной матрицы.

Таким образом, в рамках исследования инвариантов с целью выявления наиболее полного при определении изоморфизма графов заданного типа будет реализован алгоритм генерации графов с заданным числом вершин, алгоритм полиномиального времени для определения непосредственно изоморфизма графов и вычисление всех инвариантов, представленных в пунктах 1.3.1 – 1.3.8.

2 Программная реализация методов определения изоморфизма графов

2.1 Структура программы

Разработка приложения, осуществляющего автоматизацию генерации графов, вычисления их инвариантов и проверки их на изоморфизм производилась на высокоуровневом объектно-ориентированном строго типизированном языке программирования Java 8 версии [2, 15]. Также стоит отметить, что реализованное приложение является web-приложением и может быть использовано с некоторыми доработками в дальнейших исследованиях.

В процессе разработки приложения в рамках представленной работы помимо объектно-ориентированной парадигмы программирования использовалась такая схема структурирования приложения, как MVC – Model-View-Controller, или «Модель-Вид-Контроллер». Схема этого представления изображена на рисунке 15.

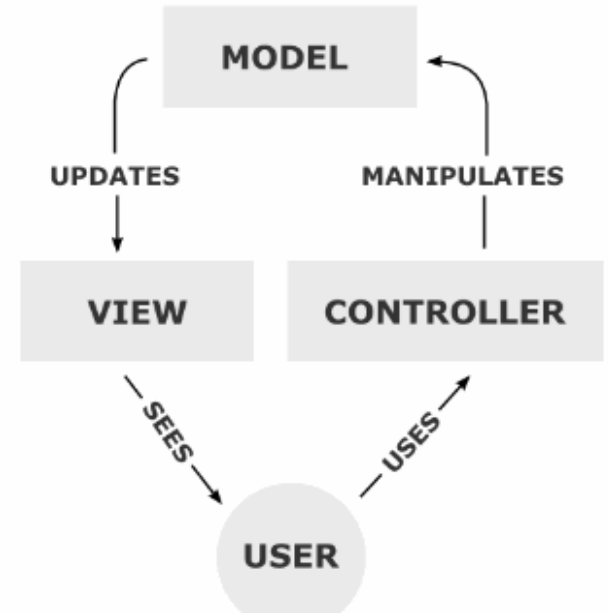


Рисунок 14 - Структура web-приложения «Модель-Вид-Контроллер»

Согласно представленному конструкционному шаблону всё приложение разделяется на 3 компонента:

– «модель» отвечает за хранение данных приложения и их обработку;

– «вид» отвечает за представление данных пользователю;

– «контроллер» отвечает за управление логикой работы приложения и ответ на взаимодействия пользователя.

Также в разработанном приложении использовались следующие программные платформы и компоненты:

– Spring Boot – программная платформа, позволяющая осуществлять быструю разработку web-приложений на основе Spring Framework. Благодаря авто-конфигурации, одной из своих отличительных особенностей, эта программная платформа позволяет настраивать специальные конфигурационные классы при помощи аннотаций.

– Apache Maven – менеджер зависимостей, позволяющий автоматизировать сборку проектов на основе их структурного описания в отдельном файле на языке POM и сформировать jar-архив приложения.

– Apache Tomcat – локальный сервер приложений, позволяющий запустить разработанное приложение на рабочем компьютере.

Разработанное в рамках данной работы приложение состоит из 12 классов:

– GenerateGraphsController, ShowMultipleResultsController, IndexController, отвечающих за управление запросами пользователя и принадлежащих компоненте «Контроллер» шаблона MVC;

– Graph, ChromaticNumberAlgorithm, EigenvaluesAlgorithm, FloydWorshellAlgorithm, PolynomialAlgorithm, отвечающих за представление графа внутри программы, вычисление его инвариантов, и принадлежащего компоненте «Модель»;

– CompareService, CounterInvariantsService, GraphGenerationService, отвечающих за дополнительный функционал приложения, включающий в

себя генерацию графов [21], сравнение значений инвариантов и подсчёт количества изоморфных графов;

– DemoApplication – главного класса приложения, запускающего его работу.

В процессе рассмотрения реализаций вычисления инвариантов остановимся на основных моментах непосредственного функционала классов в следующем подразделе.

2.2 Реализация вычисления инвариантов

Перед тем, как начать описание реализаций вычисления инвариантов графов стоит предоставить описание класса, содержащего в себе внутреннее представление графа. На рисунке 16 представлена UML-диаграмма класса Graph.

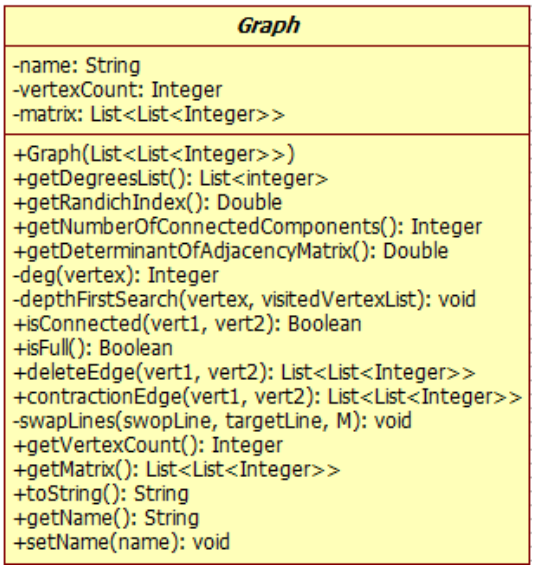


Рисунок 15 - UML-диаграмма класса Graph

Данный класс содержит в себе три закрытых поля:

– name – содержит в себе имя графа, в данной работе использовался порядковый номер при генерации;

– vertexCount – количество вершин графа, заданное пользователем;

– matrix – матрица смежности сгенерированного графа, представленная в виде списка списков из целочисленных элементов, а именно нулей и единиц.

Из наиболее ключевых методов данного класса стоит отметить следующие:

– getDegreesList() – метод, возвращающий список степеней вершин графа;

– getRandichIndex() – метод, возвращающий индекс Рандича, вычисленный по формуле (16);

– getNumberOfConnetcedComponents() – метод нахождения числа компонент связности графа, содержащий в своём теле вызов метода depthFirstSearch(vert1, visitedVertexList), реализующего обход в глубину графа, описанный в пункте 1.3.5;

– getDeterminantOdAdjacencyMatrix() – метод нахождения определителя матрицы смежности графа по алгоритму Гаусса, блок-схема которого представлена на рисунке 12.

Остальные методы используются при вычислении остальных инвариантов и имеют краткое описание в исходном коде приложения.

Инварианты индекс Винера и диаметр графа вычисляются в классе FloydWorshellAlgorithm, UML-диаграмма которого представлена на рисунке 17.

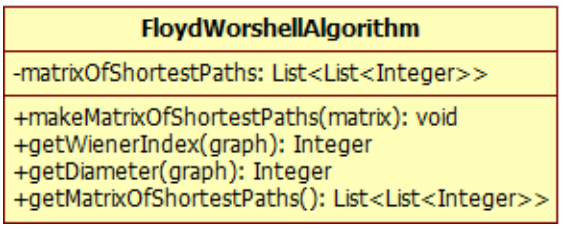


Рисунок 16 - UML-диаграмма класса FloydWorshellAlgorithm

Данный класс содержит одно закрытое поле matrixOfShortestPaths, содержащее в себе матрицу достижимости в формате списка списков из целочисленных значений, вычисленную из матрицы смежности графа в методе makeMatrixOfShortestPaths(matrix) по алгоритму Флойда-Уоршелла, блок-схема которого представлена на рисунке 10 в пункте 1.3.2.

В методе getWienerIndex(graph) производится вычисление матрицы достижимости и непосредственно значение инварианта индекса Винера по формуле (14) для графа, взятого из входного значения. В методе getDiameter(graph) также для поданного на вход графа вычисляется матрица достижимости и его диаметр.

Хроматическое число графа вычисляется в классе ChromaticNumberAlgorithm, UML-диаграмма которого представлена на рисунке 18.

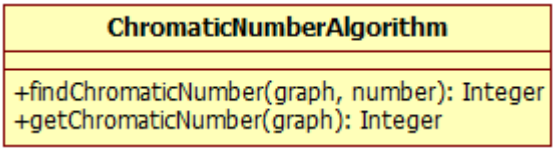


Рисунок 17 - UML-диаграмма класса ChromaticNumberAlgorithm

Данный класс не содержит полей. В методе-обёртке getChromaticNumber(graph) происходит вызов метода findChromaticNumber(graph, number), запускающего вычисление данного инварианта по теореме Зыкова, блок-схема которого представлена на рисунке 11 в пункте 1.3.6.

Вычисление упорядоченного вектора собственных чисел матрицы смежности производится в классе EigenvaluesAlgorithm, UML-диаграмма которого представлена на рисунке 19.

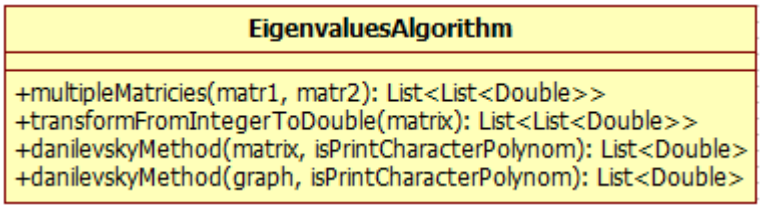


Рисунок 18 - UML-диаграмма класса EigenvaluesAlgorithm

Ключевым методом данного класса является перегруженный метод danilevskyMethod(graph, isPrintCharacterPolynom), в котором для поданного на вход графа производится вычисление собственных чисел его матрицы смежности по методу Данилевского, блок-схема которого представлена на рисунке 13 в пункте 1.3.8.

Непосредственно генерация графов с применением критерия Гавела-Хакими по алгоритму, блок-схема и описание которого представлены в пункте 1.3.1 и на рисунке 9, производится в классе GraphGenerationService, механизм сравнения инвариантов пар графов с подсчётом количества совпавших реализован в классе CountInvariantsService, классом, объединяющим данные две группы функционала, запускающим непосредственно вычисления и сбор результатов вычислений является класс CompareService.

2.3 Реализация алгоритма полиномиального времени

Алгоритм полиномиального времени, описанный в пункте 1.2.3, реализован в классе PolynomialAlgorithm, UML-диаграмма которого представлена на рисунке 20.

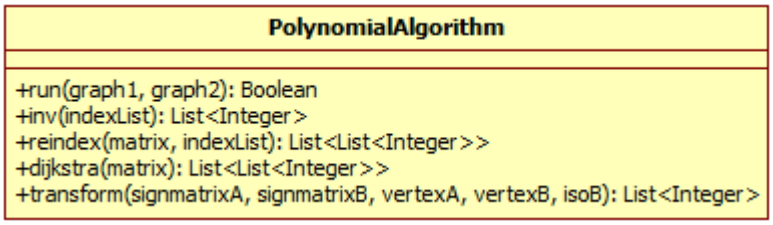


Рисунок 19 - UML-диаграмма класса PolynomialAlgorithm

Ключевым методом данного класса является метод run(graph1, graph2), в теле которого реализованы четыре процедуры, описанные в пункте 1.2.3, и непосредственно запуск самого алгоритма, блок-схемы которых представлены на рисунках 4 – 8. Данный класс так же не содержит полей, и все вычисления производятся над поступающими на вход методу run(graph1, graph2) графов с их матрицами смежности. Результатом работы алгоритма является логическое значение «истина», возвращаемое методом в случае изоморфизма двух графов и «ложь» в противном случае.

В результате, UML-диаграмма разработанного приложения принимает вид, представленный в приложении А на рисунке А.1.

2.4 Пользовательский интерфейс приложения

Разработанное web-приложение состоит из трёх web-страниц – главной страницы, страницы ввода параметров генерируемых графов и эксперимента и страницы результатов вычислений приложения.

Весь пользовательский интерфейс данного приложения был разработан с применением таких компонентов, как:

– CSS – язык стилей, позволяющий структурировать применение стилей к элементам html-страниц;

– JSP – библиотека языка Java, позволяющая внедрять исполняемый код на языке Java внутрь html-разметки страницы, тем самым, предоставляя возможность выполнять его на сервере и выдавать пользователю уже

готовую html-страницу с вычисленными данными, что позволяет создавать динамические html-страницы;

– JavaScript – язык программирования, используемый в качестве языка сценариев, позволяющий внедрить некую интерактивность на web-страницу. В данной работе использовался при построении графиков на странице результатов вычислений;

– HTML – непосредственно язык разметки страниц, основанный на языке разметки XML.

Главная страница содержит в себе кнопку перехода к вводу параметров проводимого эксперимента и может быть в дальнейшем дополнена другими кнопками по мере добавления функционала в приложение. Главная форма web-приложения представлена на рисунке А.2.

При нажатии на кнопку «Сгенерировать графы» происходит переход на страницу ввода параметров эксперимента, изображение которой представлено на рисунке А.3. На этой странице пользователю предлагается ввести количество вершин генерируемых графов и количество пар графов из общей выборки, для которых будут вычисляться инварианты и определяться изоморфизм. При вводе пользователем некорректных значений, таких как отрицательные числа, количество вершин, меньшее 3, поскольку при данном значении возможно построение только единственного варианта графа, и количество пар генерируемых графов, меньшее 1, выводится соответствующее информационное сообщение, представленное на рисунке А.4.

После выполнения всех необходимых вычислений по всем алгоритмам, пользователю откроется страница результатов экспериментов, представленная на рисунке А.5. Эта страница содержит информацию о количестве проверенных графов, таблицу с результатами сравнений значений инвариантов для пар графов с отображением числа тех, для которых значения совпали и представление данных результатов таблицы в виде столбчатой диаграммы. Также на странице представлена ещё одна столбчатая

диаграмма, отображающая разницу между реальным количеством изоморфных графов и количеством, найденным по каждому инварианту. Анализ результатов, представленных на данной странице, будет производиться в следующей главе.

Таким образом, было разработано web-приложение, позволяющее автоматизировать процесс:

– генерации заданного количества графов с заданным числом вершин;

– определения пар изоморфных графов по алгоритму полиномиального времени;

– вычисления инвариантов с подсчётом графов, для которых совпали найденные значения;

– графического представления результатов эксперимента.

3 Анализ полученных результатов

3.1 Формат проводимых экспериментов

В представленном исследовании эксперименты будут проводиться с использованием разработанного web-приложения, описанного в разделе 2. На стартовой странице запуска эксперимента, представленной в приложении А на рисунке А.2, будет выполняться ввод параметров эксперимента – количества вершин генерируемых графов и количества пар графов, которые будут взяты случайно из всего множества сгенерированных графов. Поскольку в представленном исследовании не была поставлена цель достижения наибольшей оптимизации и скорости выполнения сравнений для графов с большим количеством вершин, а главной целью является определение наиболее полного инварианта для задачи проверки наличия изоморфизма графов, было принято решение проводить исследование на графах с 4, 6 и 8 вершинами.

Так как в реализованном алгоритме генерации графов их полное количество представляет собой число всех возможных векторов со всеми возможными комбинациями степеней вершин, при дальнейшей проверке всех пар графов на предмет наличия изоморфизма приводит к большим вычислительным и временным затратам. Поэтому в рамках представленного исследования было принято решение ввести такой параметр, как количество пар генерируемых графов, которые будут участвовать в дальнейших вычислениях инвариантов и проверке их на изоморфизм.

Таким образом, эксперименты будут проводиться на выборках в 50, 150, 300 и 500 пар графов с 4, 6 и 8 вершинами с подсчётом общего числа вошедших в эти пары графов. По каждому инварианту будет выполняться подсчёт количества графов с равными значениями инвариантов, которое затем будет сравниваться с числом изоморфных графов, вычисленным по алгоритму полиномиального времени.

3.2 Результаты экспериментов

Все эксперименты проводились последовательно для 50, 150, 300 и 500 пар графов сначала с 4, затем с 6, затем с 8 вершинами. Результаты были сведены в 3 таблицы, в которые было занесено число изоморфных графов по каждому инварианту для указанного числа пар графов в выборке. На основании полученных результатов таблиц были построены графики, отображающие разницу между значением количества изоморфных графов по инварианту и количеством изоморфных графов определённому по алгоритму полиномиального времени.

В таблице 1 представлены результаты экспериментов для графов с 4 вершинами.

Таблица 1 – Результаты экспериментов для графов с 4 вершинами

Инвариант	Количество пар графов			
	50	150	300	500
Вектор степеней вершин	14	50	82	140
Индекс Виннера	16	60	96	154
Индекс Рандича	14	52	86	146
Диаметр графа	60	134	258	342
Число компонент связности	94	230	422	556
Хроматическое число	48	110	200	280
Определитель матрицы смежности	24	72	144	180
Вектор собственных чисел матрицы смежности	14	50	82	140
Алгоритм полиномиального времени	14	50	82	140
Всего графов в выборке	98	250	444	596

На основании выше представленной таблицы был построен график разностей между количеством изоморфных графов, вычисленным по алгоритму полиномиального времени, и количеством изоморфных графов,

вычисленным по каждому инварианту для графов с 4 вершинами. График представлен на рисунке 20.

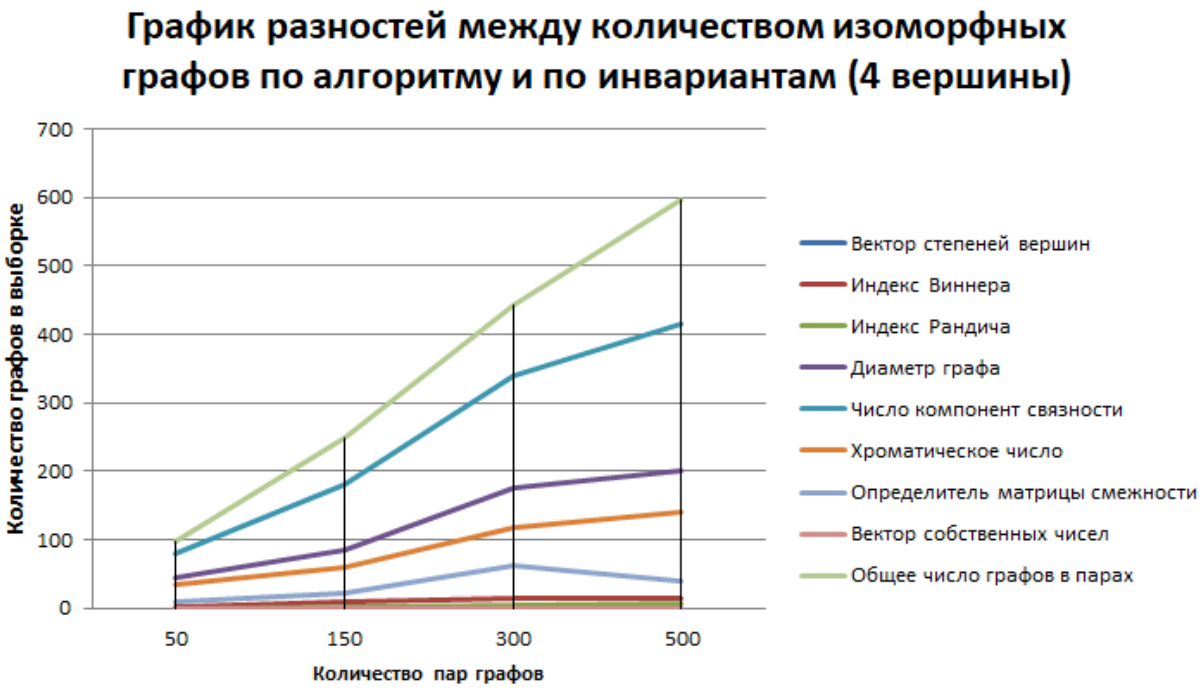


Рисунок 20 – График разностей между количеством изоморфных графов с 4 вершинами по алгоритму полиномиального времени и по инвариантам

В таблице 2 представлены результаты экспериментов для графов с 6 вершинами.

Таблица 2 – Результаты экспериментов для графов с 6 вершинами

Инвариант	Количество пар графов			
	50	150	300	500
Вектор степеней вершин	2	8	16	22
Индекс Виннера	12	46	94	124
Индекс Рандича	2	4	2	6
Диаметр графа	70	134	338	604
Число компонент связности	100	284	552	942

Продолжение таблицы 2 – Результаты экспериментов для графов с 6 вершинами

Инвариант	Количество пар графов			
	50	150	300	500
Хроматическое число	56	140	250	424
Определитель матрицы смежности	56	152	296	446
Вектор собственных чисел матрицы смежности	2	8	16	22
Алгоритм полиномиального времени	2	8	16	22
Всего графов в выборке	100	290	556	952

На основании выше представленной таблицы был построен график разностей между количеством изоморфных графов, вычисленным по алгоритму полиномиального времени, и количеством изоморфных графов, вычисленным по каждому инварианту для графов с 6 вершинами. График представлен на рисунке 21.

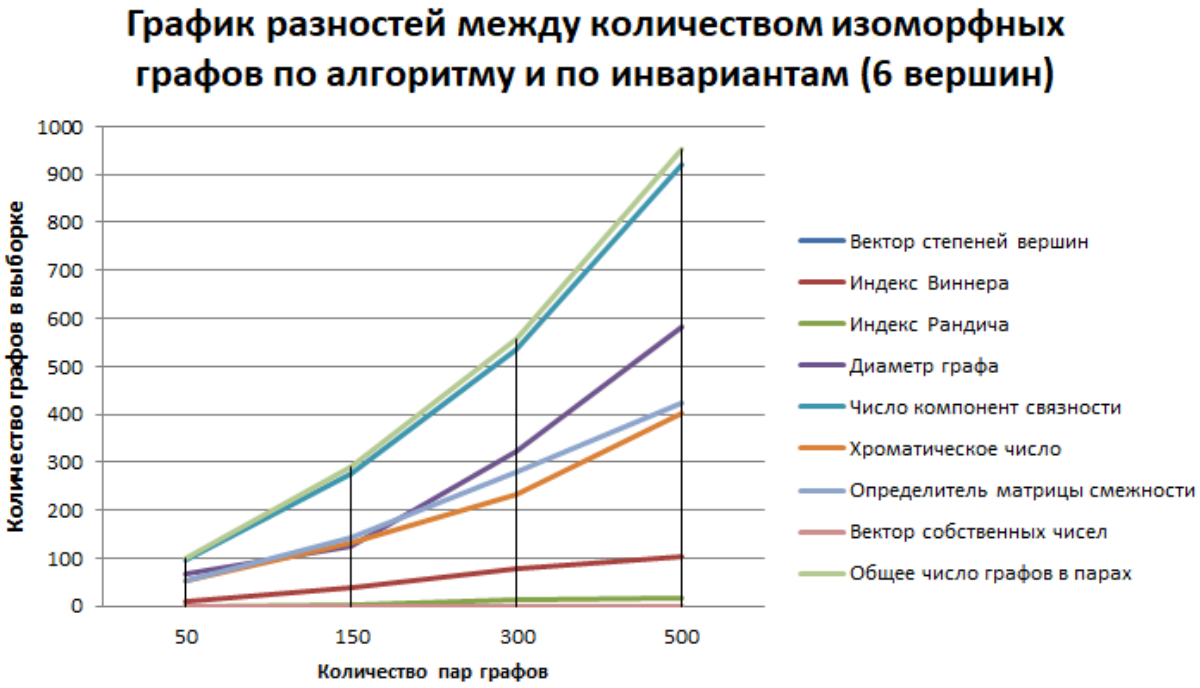


Рисунок 21 – График разностей между количеством изоморфных графов с 6 вершинами по алгоритму полиномиального времени и по инвариантам

В таблице 3 представлены результаты экспериментов для графов с 8 вершинами.

Таблица 3 - Результаты экспериментов для графов с 8 вершинами

Инвариант	Количество пар графов			
	50	150	300	500
Вектор степеней вершин	0	4	2	4
Индекс Виннера	4	34	82	64
Индекс Рандича	0	4	0	4
Диаметр графа	54	188	356	574
Число компонент связности	96	292	588	982
Хроматическое число	50	150	298	492
Определитель матрицы смежности	44	158	282	504
Вектор собственных чисел матрицы смежности	0	2	2	2
Алгоритм полиномиального времени	0	2	2	2
Всего графов в выборке	100	300	596	990

На основании выше представленной таблицы был построен график разностей между количеством изоморфных графов, вычисленным по алгоритму полиномиального времени, и количеством изоморфных графов, вычисленным по каждому инварианту для графов с 8 вершинами. График представлен на рисунке 22.

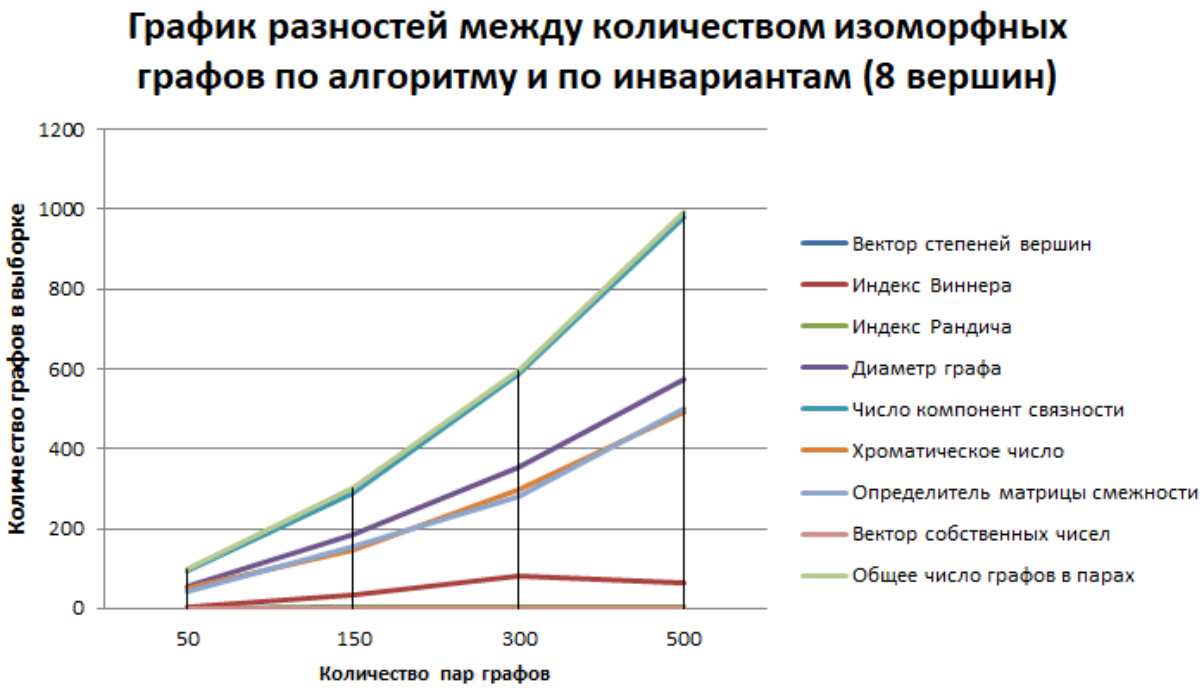


Рисунок 22 – График разностей между количеством изоморфных графов с 8 вершинами по алгоритму полиномиального времени и по инвариантам

Анализ всех полученных результатов экспериментов будет представлен в следующем подразделе.

3.3 Сравнение инвариантов графа

Проведём анализ полученных результатов экспериментов из подраздела 3.2.

Возвращаясь к предположению, выдвинутому в конце пункта 1.3.7, стоит отметить, что при вычислении определителей матриц смежности графов более чем в половине случаев не удавалось произвести расчёты этого инварианта, так как вид графов, участвующих в исследовании не подразумевает наличие петель, и, следовательно, отличных от нуля элементов на главной диагонали. В связи с этим, возникают случаи, когда не удаётся найти такую перестановку столбцов матрицы, при которой было бы возможным преобразовать исходную матрицу к треугольному виду.

Следовательно, можно сделать вывод о низкой эффективности применения этого инварианта при определении изоморфизма графов.

Также низкие результаты эффективности наблюдаются у таких инвариантов, как число компонент связности, диаметр и хроматическое число графа. Как видно из рисунков 20 – 22, для этих инвариантов значение разности между действительным числом изоморфных графов и числом, определённым по вычисленным значениям инвариантов приближается к общему числу графов выборки. При высокой эффективности инварианта это число должно стремиться к нулю, означая, что разница между действительным числом изоморфных графов и числом, полученным по инварианту, если не равна нулю, то должна быть близка к нулю, свидетельствуя о равных значениях.

Инварианты индекс Виннера и индекс Рандича, исходя из тех же рисунков графиков, имеют меньшие значения разностей, близкие к нулю, но только в случае графов с 4 вершинами. В случае 8 вершин в процессе экспериментов можно было наблюдать, наоборот, заниженное число найденных изоморфных графов по индексу Рандича по сравнению с действительным. Например, как видно из таблицы 3, на наборе из 300 пар графов, в которые входило 596 различных графов, согласно индексу Рандича не было найдено ни одного изоморфного графа, хотя по алгоритму полиномиального времени была найдена одна пара. Стоит отметить, что инварианты диаметр графа и число компонент связности, напротив, выдавали завышенное число изоморфных графов, превышающее действительное, и составляя в среднем 55% и 90% от общего числа графов в выборке. Согласно хроматическому числу также было выявлено больше изоморфных графов, и их число составляет в среднем 47% от общего числа графов в выборке. В случае индекса Виннера – превышение на 15% от общего числа графов в выборке.

Индекс Виннера и индекс Рандича не являются достаточно эффективными инвариантами в решении задачи определения изоморфизма

графов, но они могут быть использованы в совокупности с другими инвариантами, к примеру, при первоначальной проверке графов. Окончательное решение об изоморфизме графов в данном случае может быть принято на основании других инвариантов.

Вектор степеней вершин имеет высокую эффективность при определении изоморфизма графов с 4 и 6 вершинами. В этом случае разница между реальным количеством инвариантов и количеством, определённым по равенству векторов степеней вершин графов, равна 0 для всех рассматриваемых объёмов выборок. В экспериментах с 8 вершинами на наборах в 150 и 500 пар графов наблюдаются 2 излишне выбранных графа. Исходя из этого, можно сделать вывод о том, что уже в случае 8 вершин может наблюдаться тот случай, в котором неизоморфные графы могут иметь идентичные вектора степеней вершин. Однако этот инвариант, как и в случае с индексом Виннера и индексом Рандича может быть использован совместно с другими инвариантами.

Из всех рассмотренных инвариантов графов наиболее полным является вектор собственных чисел матрицы смежности графа. Исходя из таблиц 1 – 3 по этому инварианту было вычислено равное количество изоморфных графов с количеством, полученным по алгоритму полиномиального времени. На рисунках 20 – 22 график разностей также совпадает с нулевой осью.

Таким образом, в результате анализа проведённых вычислительных экспериментов можно сделать вывод о том, что для графов без петель и кратных рёбер с количеством вершин меньшим или равным 8 наиболее полным инвариантом считается вектор собственных чисел матрицы смежности. Также имеют место быть применёнными в совокупности такие инварианты, как вектор степеней вершин графа, индекс Виннера, индекс Рандича. Наименее эффективными оказались такие инварианты, как диаметр графа, определитель матрицы смежности, число компонент связности, хроматическое число.

Заключение

В ходе выполнения выпускной квалификационной работы было проведено исследование, объектом которого являлась задача определения изоморфизма графов с применением подхода, заключающегося в отыскании наиболее полного инварианта, из совпадения которого следовал бы изоморфизм графов. Были реализованы на языке Java алгоритм полиномиального времени и вычисление таких инвариантов, как:

- вектор степеней вершин графа;
- индекс Виннера;
- индекс Рандича;
- диаметр графа;
- число компонент связности;
- хроматическое число;
- определитель матрицы смежности;
- вектор собственных чисел атрицы смежности.

В ходе работы были выполнены следующие задачи:

1. Рассмотрены существующие алгоритмы решения задачи определения изоморфизма графов;
2. Рассмотрены инварианты графов;
3. Реализован алгоритм полиномиального времени, относящийся к группе методов, основанных на полном переборе и его модификациях, при решении задачи определения изоморфизма графов;
4. Реализованы вычисления инвариантов;
5. Реализован автоматизированный подход к проведению экспериментов, заключающийся в объединении процесса генерации графов с заданными параметрами и проведения проверки их на изоморфизм;
6. Проведены эксперименты и анализ полученных результатов.

Согласно анализу, наиболее полным инвариантом при определении изоморфизма графов описанного в представленной работе вида является вектор собственных чисел матрицы смежности. Показатели, свидетельствующие о наименьшей эффективности применения инварианта при определении изоморфизма графов, были зафиксированы у числа компонент связности, диаметра графа и определителя матрицы смежности.

Список используемой литературы

1. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Численные методы / Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков – М.: Лаборатория знаний. Бином, 2019. – 636 с.;
2. Блох Дж. Java. Эффективное программирование / Дж. Блох – М.: Вильямс, 2018. – 464 с.;
3. Андреев А. Е., Болотов А. А., Коляда К. В., Фролов А. Б. Дискретная математика. Прикладные задачи и сложность алгоритмов. Учебник и практикум для академического бакалавриата / А. Е. Андреев, А. А. Болотов, К. В. Коляда, А. Б. Фролов – М.: Юрайт, 2017. – 317 с.;
4. Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И. Лекции по теории графов / В.А. Емеличев, О. И. Мельников О. И., В. И. Сарванов, Р. И. Тышкевич – М.: Ленанд, 2017. – 390 с.;
5. Зализняк В. Е. Численные методы. Основы научных вычислений / В. Е. Зализняк – М.: Юрайт, 2020 – 357 с.;
6. Земляченко В. Н., Корнеенко Н. М., Тышкевич Р. И. Проблема изоморфизма графов // Теория сложности вычислений, I. Записки научных семинаров ЛОМИ. - 1982. - Т.118. - С.83-158.;
7. Харари Ф. Теория графов / Ф. Харари – М.: Ленанд, 2018. – 304 с.;
8. Рафгарден Т. Совершенные алгоритмы. Графовые алгоритмы и структуры данных / Т. Рафгарден – С.-П.: Питер, 2019. – 256 с.;
9. Струченков В. И. Прикладные задачи оптимизации. Модели, методы, алгоритмы / В. И. Струченков – М.: Солон, 2016. – 314 с.;
10. Оре О. Графы и их применение / О. Оре – М.: Ленанд, 2015. – 208 с.;
11. Пантелеев А. В., И. А. Кудрявцева Численные методы. Практикум. Учебное пособие / А. В. Пантелеев, И. А. Кудрявцева – М.: Инфра-М, 2017. – 512 с.;

12. Понтрягин Л. С. – Алгебра. Теория определителей. Корни многочленов и комплексные числа. Приведение матриц к каноническому виду / Л. С. Понтрягин – М.: Едиториал УРСС, 2018. – 136 с.;

13. Седжвик Р. Алгоритмы на C++. Анализ структуры данных. Сортировка. Поиск. Алгоритмы на графах. Руководство / Р. Седжвик – М.: Вильямс, 2019. – 1056 с.;

14. Уилсон Р. Введение в теорию графов / Р. Уилсон – М.: Вильямс, 2020. – 240 с.;

15. Эккель Б. Философия Java / Б. Эккель – С.-П.: Питер, 2019. – 1168 с.;

16. Cordella L, Foggia P, Sansone C, Vento M. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs // IEEE Transactions on Pattern Analysis and Machine Intelligence – 2004. – 26 (10). – P. 1367-1372;

17. Dharwadker A., Tevet J. The Graph Isomorphism Algorithm: Graph Isomorphism is in P // A. Dharwadker, J. Tevet. – 2011. – 10. – P. 38;

18. Erdos, P. A Simple Havel-Hakimi Type Algorithm to Realize Graphical Degree Sequences of Directed Graphs / P. Erdos, I. Miklos, Z. Toroczkai // Electr. J. Comb. – 2010. – V. 17, № 1. – P. 1-10.

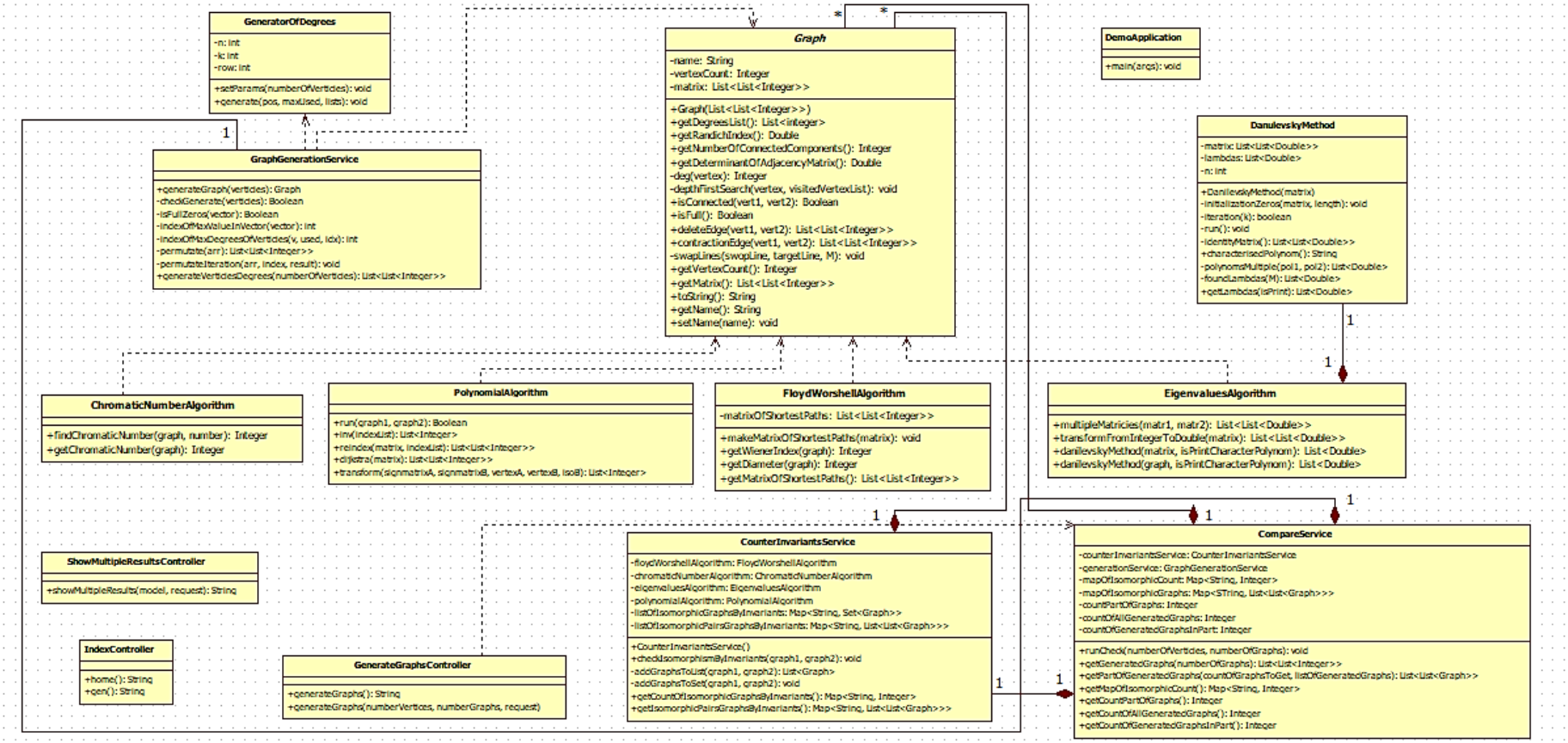
19. Ullmann Julian R. An algorithm for Subgraph Isomorphism // Journal of the Association for Computing Machinery – 1976. – 23. – P. 31-42;

20. Ullmann Julian R. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism // J Exp Algorithmics. – 2011. – 15 (1.6) – P. 1-7;

21. Blitzstein J., Diaconis P. A Sequential Importance Sampling Algo-rithm for Generating Random Graphs with Prescribed Degrees [Электронный ресурс]. – Режим доступа: https://people.fas.harvard.edu/~blitz/BlitzsteinDiaconisGraph-Algorithm.pdf.

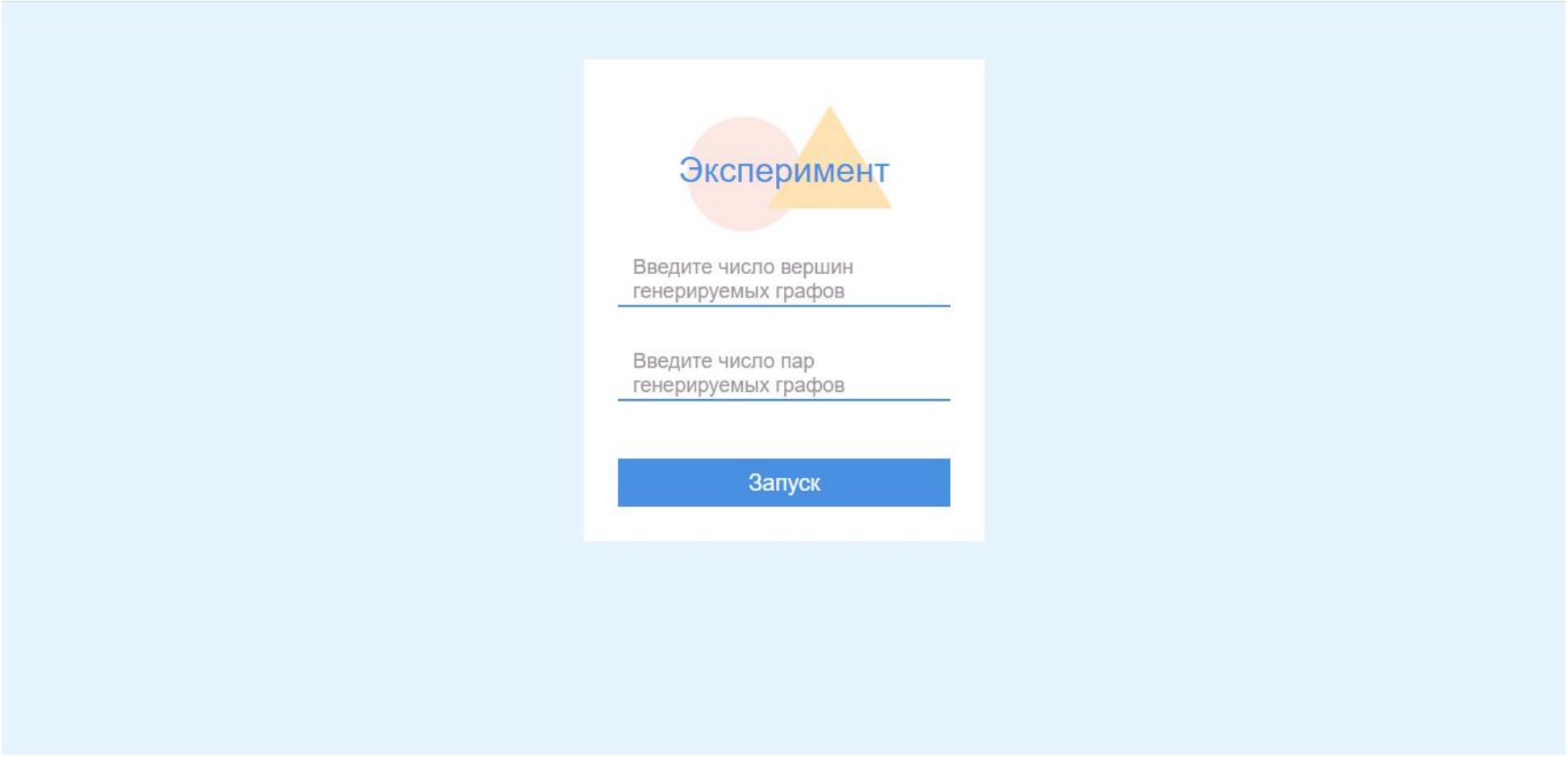
Приложение А

Иллюстрационный материал к выпускной квалификационной работе

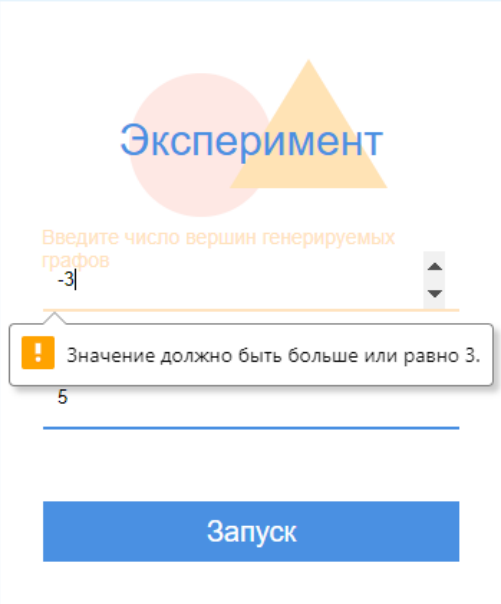


Приложение А. 1 – UML-диаграмма разработанного web-приложения

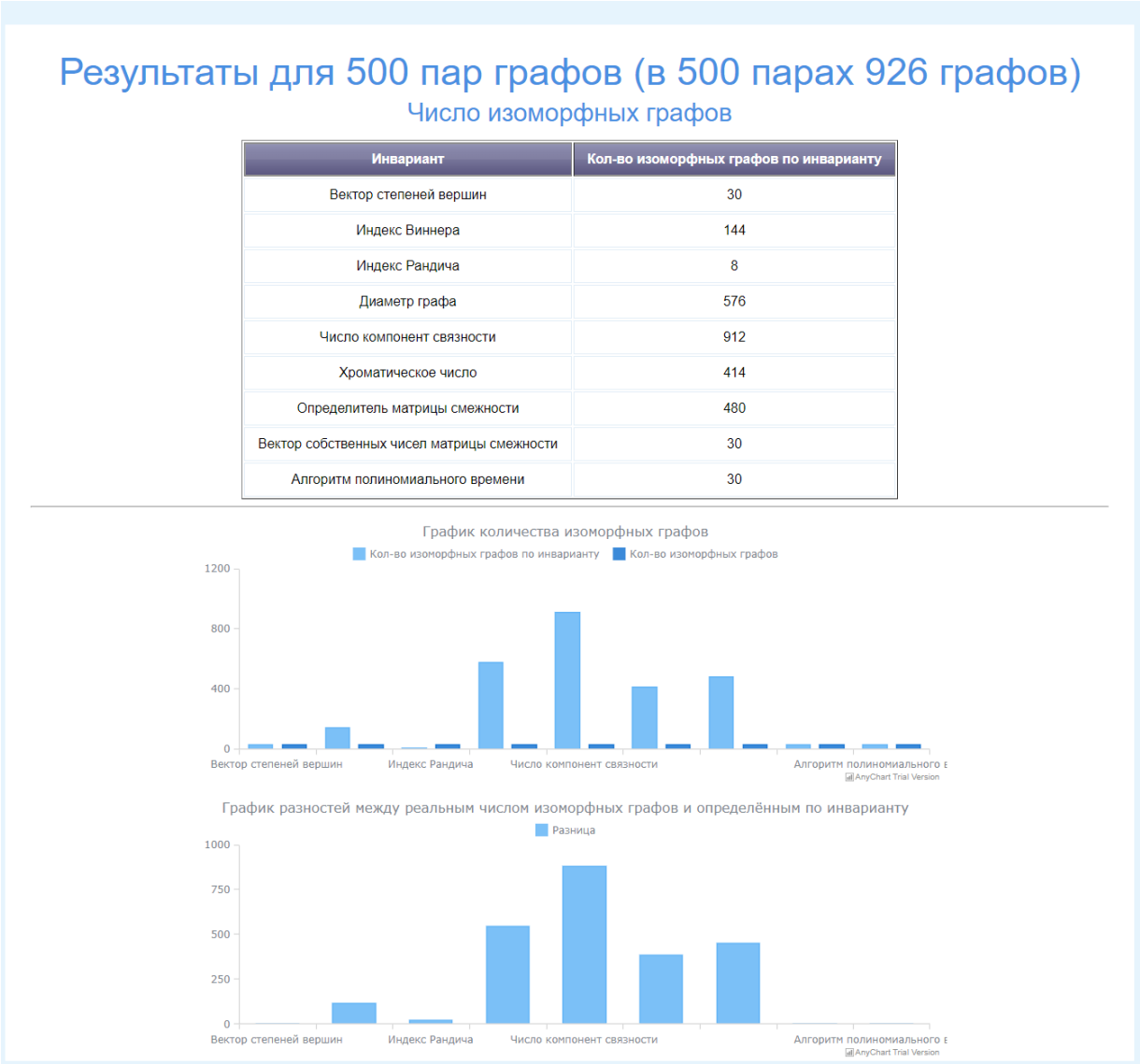
Приложение А. 2 – Главная страница приложения



Приложение А. 3 – Страница генерации графов и запуска эксперимента



Приложение А. 4 – Ответ страницы на некорректно введённые данные



Приложение А. 5 – Фрагмент страницы результатов вычислений с таблицей