ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(НИУ «БелГУ»)

ФАКУЛЬТЕТ МАТЕМАТИКИ И ЕСТЕСТВЕННОНАУЧНОГО ОБРАЗОВАНИЯ ПЕДАГОГИЧЕСКОГО ИНСТИТУТА

КАФЕДРА ИНФОРМАТИКИ, ЕСТЕСТВЕННОНАУЧНЫХ ДИСЦИПЛИН И МЕТОДИК ПРЕПОДАВАНИЯ

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ ТЕОРИИ ГРАФОВ

Выпускная квалификационная работа
обучающегося по направлению подготовки 44.03.05 Педагогическое образование, профиль информатика и иностранный язык
очной формы обучения, группы 02041205
Ракитина Романа Евгеньевича

Научный руководитель:
к. ф.-м. н., доцент Старовойтов А.С.

БЕЛГОРОД 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Когда людей спрашивают, что такое граф, большинство людей представляют график, или диаграмму, которая показывает производственную деятельность какого-нибудь предприятия, или свойства какой-нибудь математической формулы.

Но для людей, которые имеют дело и математикой, слово «граф» имеет совсем другое значение.

Начало теории графов как математической дисциплине было положено Эйлером в его знаменитом рассуждении о Кёнигсбергских мостах. Однако, эта статья Эйлера 1736 года была единственной в течение почти ста лет. Интерес к проблемам теории графов возродился около середины прошлого столетия и был сосредоточен, главным образом в Англии. Имелось много причин для такого оживления изучения графов. Естественные науки оказали свое влияние на это, благодаря исследованиям электрических сетей, моделей кристаллов и структур молекул. Развития формальной логики привело к изучению бинарных отношений в форме графов. Большое число популярных головоломок поддавалось формулировкам непосредственно в терминах графов, и это приводило к пониманию, что многие задачи такого рода содержат некоторое математическое ядро, важность которого выходит за рамки конкретного вопроса. Наиболее знаменитая среди этих задач - проблема четырех красок, впервые поставленная перед математиками Де Морганом около 1850 года. Никакая другая проблема не вызывала столь многочисленных и остроумных работ в области теории графов. Благодаря своей простой формулировке и раздражающей неуловимости она до сих пор остается мощным стимулом исследований различных свойств графов.

Настоящее столетие было свидетелем неуклонного развития теории графов, которая за последние десять лет и даже двадцать вступила в новый период интенсивных разработок. В этом процессе явно заметно влияние

запросов новых областей приложений: теории игр и программирования, теории передачи сообщений, электрических сетей и контактных цепей, а также проблем биологии и психологии.

Данная работа состоит из трех глав.

В первой главе демонстрируется начальные этапы развития графов.

Вторая глава включает в себя описание алгоритмов и их применение.

Третья глава содержит код программы с детальным описание функций и формул, которые были использованы.

Для реализации программного комплекса использовался язык высокого уровня Delphi и среда разработки Borland Delphi 2007. Этот выбор обусловлен исходя из следующих критериев:

1. Среда разработки и язык позволяют быстро и качественно создавать пользовательский интерфейс высокого уровня, используя дизайнер форм, сводя к минимуму труд программиста.

2. Язык Delphi так же обладает рядом преимуществ по сравнению с Object Pascal, т.к. является более новым и более развитым языком.

3. В нём на очень высоком уровне реализованы механизмы безопасности кода.

4. Ещё одним несомненным плюсом является наличие русскоязычной развитой справочной системы, что значительно упрощает процесс программирования.

Цель работы: разработать приложение для решения задач с использованием теории графов.

Достижение поставленной цели предполагает выполнение следующих задач:

- изучить предметную область;
- проанализировать методы решения поставленной задачи;
- осуществить проектирование программного продукта;
- выбрать средства и технологии разработки;

- разработать программный продукт;
- провести комплексное тестирование программного продукта;

Объект исследования – решения задач с использованием теории графов.

Предмет исследования – приложение для решения задач с использованием теории графов.

1. НАЧАЛЬНЫЕ ЭТАПЫ РАЗВИТИЯ ГРАФОВ

1.1 Историческая справка

Теория графов - это область дискретной математики, особенностью которой является геометрический подход к изучению объектов. Теория графов находится сейчас в самом расцвете. Обычно её относят к топологии (потому что во многих случаях рассматриваются лишь топологические свойства графов), однако она пересекается со многими разделами теории множеств, комбинаторной математики, алгебры, геометрии, теории матриц, теории игр, математической логики и многих других математических дисциплин. Основной объект теории графов-граф и его обобщения [1].

Первые задачи теории графов были связаны с решением математических развлекательных задач и головоломок (задача о Кенигсбергских мостах, задача о расстановке ферзей на шахматной доске, задачи о перевозках, задача о кругосветном путешествии и другие). Одним из первых результатов в теории графов явился критерий существования обхода всех ребер графа без повторений, полученный Л. Эйлером при решении задачи о Кенигсбергских мостах. Вот пересказ отрывка из письма Эйлера от 13 марта 1736 году: «Мне была предложена задача об острове, расположенном в городе Кенигсберге и окруженном рекой, через которую перекинуто 7 мостов. Спрашивается, может ли кто-нибудь непрерывно обойти их, проходя только однажды через каждый мост. И тут же мне было сообщено, что никто еще до сих пор не смог это проделать, но никто и не доказал, что это невозможно. Вопрос этот, хотя и банальный, показался мне, однако, достойным внимания тем, что для его решения недостаточны ни геометрия, ни алгебра, ни комбинаторное искусство. После долгих размышлений я нашел лёгкое правило, основанное на вполне убедительном доказательстве, с помощью которого можно во всех задачах такого рода

тотчас же определить, может ли быть совершен такой обход через какое угодно число и как угодно расположенных мостов или не может» [2].

Кенигсбергские мосты схематически можно изобразить так (см. рисунок 1):

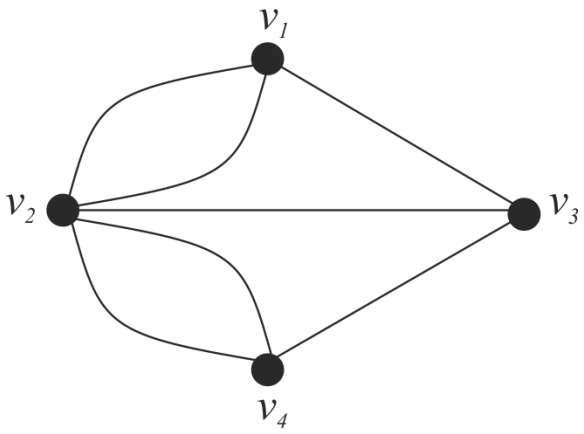

Рисунок 1 – Схема Кенигсбергских мостов

Правило Эйлера:

1. В графе, не имеющем вершин нечетных степеней, существует обход всех рёбер (причем каждое ребро проходится в точности один раз) с началом в любой вершине графа.

2. В графе, имеющем две и только две вершины с нечетными степенями, существует обход с началом в одной вершине с нечетной степенью и концом в другой.

3. В графе, имеющим более двух вершин с нечетной степенью, такого обхода не существует [3].

Существует еще один вид задач, связанных с путешествиями вдоль графов. Речь идёт о задачах, в которых требуется отыскать путь, проходящий через все вершины, причем не более одного раза через каждую. Цикл, проходящий через каждую вершину один и только один раз, носит название гамильтоновой линии (в честь Уильяма Роуэна Гамильтона, знаменитого ирландского математика прошлого века, который первым начал изучать

такие линии). К сожалению, пока еще не найден общий критерий, с помощью которого можно было бы решить, является ли данный граф гамильтоновым, и если да, то найти на нём все гамильтоновы линии.

Сформулированная в середине 19 в. проблема четырех красок также выглядит как развлекательная задача, однако попытки ее решения привели к появлению некоторых исследований графов, имеющих теоретическое и прикладное значение. Проблема четырех красок формулируется так: «Можно ли область любой плоской карты раскрасить четырьмя цветами так, чтобы любые две соседние области были раскрашены в различные цвета?». Гипотеза о том, что ответ утвердительный, была сформулирована в середине 19в. В 1890 году было доказано более слабое утверждение, а именно, что любая плоская карта раскрашивается в пять цветов. Сопоставляя любой плоской карте двойственный ей плоский граф, получают эквивалентную формулировку задачи в терминах графов: Верно ли, что хроматическое число любого плоского графа меньше либо равно четырёх? Многочисленные попытки решения задачи оказали влияние на развитие ряда направлений теории графов. В 1976 году анонсировано положительное решение задачи с использованием ЭВМ [3].

Другая старая топологическая задача, которая особенно долго не поддавалась решению и будоражила умы любителей головоломок, известна как «задача об электро, газо и водоснабжении». В 1917 году Генри Э. Дьюдени дал ей такую формулировку. В каждый из трёх домов (см. рисунок 2) необходимо провести газ, свет и воду.

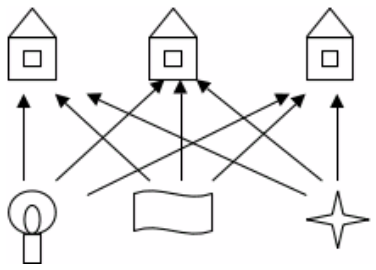

Рисунок 2 – Свет, вода, газ

Можно ли так проложить коммуникации, чтобы они, нигде не пересекаясь друг с другом, соединяли каждый дом с источниками электричества, газа и воды? Иначе говоря, можно построить плоский граф с вершинами в шести указанных точках? Оказывается, такой граф построить нельзя. Об этом говорится в одной очень важной теореме – так называемой теореме Куратовского. Теорема утверждает, что каждый граф, не являющийся плоским, содержит в качестве подграфа один из двух простейших пространственных графов (см. рисунок 3).

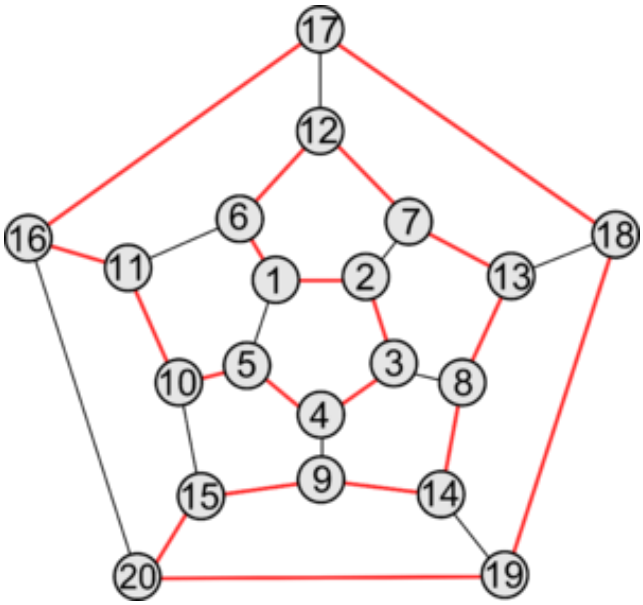

Рисунок 3 – Теорема Куратовского

В середине 19 в. появились работы, в которых при решении практических задач были получены результаты, относящиеся к теории графов.

Так, например, Г. Кирхгоф при составлении полной системы уравнений для токов и напряжений в электрической схеме предложил по существу представлять такую схему графом и находить в этом графе островные деревья, с помощью которых выделяются линейно независимые системы

контуров. А. Кэли, исходя из задач подсчета числа изомеров предельных углеводородов, пришел к задачам перечисления и описания деревьев, обладающих заданными свойствами, и решил некоторые из них [4].

В 20 в. задачи, связанные с графами, начали возникать не только в физике, химии, электротехнике биологии, экономике, социологии и т.д., но и внутри математики, в таких разделах, как топология, алгебра, теория вероятностей, теория чисел. В начале 20 в. графы стали использоваться для представления некоторых математических объектов и формальной постановки различных дискретных задач; при этом наряду с термином «граф» употреблялись и другие термины, например, карта, комплекс, диаграмма, сеть, лабиринт. После выхода в свет в 1936 году монографии Д. Кёнига термин «граф» стал более употребительным, чем другие. В этой работе были систематизированы известные к тому времени факты. В 1936 году вышла небольшая брошюра Ойстена Оре, содержащая блестящее элементарное введение в теорию графов. В 1962 году в Англии была издана книга французского математика Клода Бержа “Теория графов и её приложение”. Обе книги, безусловно, представляют интерес для любителей занимательной математики. Сотни известных головоломок, на первый взгляд не имеющих ничего общего друг с другом, легко решаются с помощью теории графов [5].

В 20-30-х годах 20 в. появились первые результаты, относящиеся к изучению свойств связности, планарности, симметрии графов, которые привели к формированию ряда новых направлений в теории графов.

Значительно расширились исследования по теории графов в конце 40-х - начале 50-х годов, прежде всего в силу развития кибернетики и вычислительной техники. Благодаря развитию вычислительной техники, изучению сложных кибернетических систем, интерес к теории графов возрос, а проблематика теории графов существенным образом обогатилась. Кроме того, использование ЭВМ позволило решать возникающие на практике

конкретные задачи, связанные с большим объемом вычислений, прежде не поддававшиеся решению. Для ряда экстремальных задач теории графов были разработаны методы их решения, например, один из таких методов позволяет решать задачи о построении максимального потока через сеть. Для отдельных классов графов (деревья, плоские графы и т. д.), которые изучались и ранее, было показано, что решения некоторых задач для графов из этих классов находятся проще, чем для произвольных графов (нахождение условий существования графов с заданными свойствами, установление изоморфизма графов и др.) [6].

Характеризуя проблематику теории графов, можно отметить, что некоторые направления носят более комбинаторный характер, другие - более геометрический. К первым относятся, например, задачи о подсчете и перечислении графов с фиксированными свойствами, задачи о построении графов с заданными свойствами. Геометрический (топологический) характер носят многие циклы задач теории графов, например, графов обходы, графов укладки. Существуют направления, связанные с различными классификациями графов, например, по свойствам их разложения.

Примером результата о существовании графов с фиксированными свойствами может служить критерий реализуемости чисел степенями вершин некоторого графа: набор целых чисел, сумма которых четна, можно реализовать степенями вершин графа без петель и кратных ребер тогда и только тогда, когда для любого r выполняется условие [7].

Примерами задач о подсчете графов с заданными свойствами являются задачи о нахождении количеств неизоморфных графов с одинаковым числом вершин и (или) ребер. Для числа неизоморфных деревьев с n вершинами была получена асимптотическая формула где C= 0,534948..., e= 2,95576...

Наряду с проблемами, носящими общий характер, в теории графов имеются специфические циклы задач. В одном из них изучаются различные свойства связности графов, исследуется строение графов по свойствам

связности. При анализе надежности сетей связи, электронных схем, коммуникационных сетей возникает задача о нахождении количеств непересекающихся цепей, соединяющих различные вершины графа. Здесь получен ряд результатов. Например, наименьшее число вершин, разделяющих две несмежные вершины графа, равно наибольшему числу непересекающихся (по вершинам) простых цепей, соединяющих эту пару вершин. Найдены критерии и построены эффективные алгоритмы установления меры связности графа (наименьшего числа вершин или ребер, удаление которых нарушает связность графа) [8].

В другом направлении исследований теории графов изучаются маршруты, содержащие все вершины или ребра графа. Известен простой критерий существования маршрута, содержащего все ребра графа: в связном графе цикл, содержащий все ребра и проходящий по каждому ребру один раз, существует тогда и только тогда, когда все вершины графа имеют четные степени. В случае обхода множества вершин графа имеется только ряд достаточных условий существования цикла, проходящего по всем вершинам графа по одному разу. Характерным специфическим направлением теории графов является цикл задач, связанный с раскрасками графов, в котором изучаются разбиения множества вершин (ребер), обладающие определенными свойствами, например, смежные вершины (ребра) должны принадлежать различным множествам (вершины или ребра из одного множества окрашиваются одним цветом). Было доказано, что наименьшее число цветов, достаточное для раскраски ребер любого графа без петель с максимальной степенью а, равно 3*а/2, а для раскраски вершин любого графа без петель и кратных ребер достаточно а+1 цветов [9].

Существуют и другие циклы задач, некоторые из них сложились под влиянием различных разделов математики. Так, под влиянием топологии производится изучение вложений графов в различные поверхности. Например, было получено необходимое и достаточное условие вложения

графа в плоскость (критерий Понтрягина - Куратовского см. выше): граф является плоским тогда и только тогда, когда он не содержит подграфов, получаемых с помощью подразбиения ребер из полного 5-вершинного графа и полного двудольного графа с тремя вершинами в каждой доле. Под влиянием алгебры стали изучаться группы автоморфизмов графов. В частности, было доказано, что каждая конечная группа изоморфна группе автоморфизмов некоторого графа. Влияние теории вероятностей сказалось на исследовании графов случайных. Многие свойства были изучены для «почти всех» графов; например, было показано, что почти все графы с n вершинами связаны, имеют диаметр 2, обладают гамильтоновым циклом (циклом, проходящим через все вершины графа по одному разу) [10].

В теории графов существуют специфические методы решения экстремальных задач. Один из них основан на теореме о максимальном потоке и минимальном разрезе, утверждающей, что максимальный поток, который можно пропустить через сеть из вершины U в вершину V, равен минимальной пропускной способности разрезов, разделяющих вершины U и V. Были построены различные эффективные алгоритмы нахождения максимального потока.

Большое значение в теории графов имеют алгоритмические вопросы. Для конечных графов, т. е. для графов с конечным множеством вершин и ребер, как правило, проблема существования алгоритма решения задач, в том числе экстремальных, решается положительно. Решение многих задач, связанных с конечными графами, может быть выполнено с помощью полного перебора всех допустимых вариантов. Однако таким способом удается решить задачу только для графов с небольшим числом вершин и ребер. Поэтому существенное значение для теории графов имеет построение эффективных алгоритмов, находящих точное или приближенное решение. Для некоторых задач такие алгоритмы построены, например, для

установления планарности графов, определения изоморфизма деревьев, нахождения максимального потока.

Результаты и методы теории графов применяются при решении транспортных задач о перевозках, для нахождения оптимальных решений задачи о назначениях, для выделения «узких мест» при планировании и управлении разработок проектов, при составлении оптимальных маршрутов доставки грузов, а также при моделировании сложных технология, процессов, в построении различных дискретных устройств, в программировании и т. д.

1.2 Основные термины и теоремы теории графов

Граф - Пара объектов G = (X, Г), где X - конечное множество, а Г – конечное подмножество прямого произведения X*X. При этом X называется множеством вершин, а Г - множеством дуг графа G. Любое конечное множество точек (вершин), некоторые из которых попарно соединены стрелками, (в теории графов эти стрелки называются дугами), можно рассматривать как граф. Если в множестве Г все пары упорядочены, то такой граф называют ориентированным. Дуга- ребро ориентированного графа. Граф называется вырожденным, если у него нет рёбер. Вершина X называется инцидентной ребру G, если ребро соединяет эту вершину с какой-либо другой вершиной. Подграфом G (V1, E1) графа G (V, E) называется граф с множеством вершин V1-Vx и множеством ребер (дуг) E1-Ex, - такими, что каждое ребро (дуга) из E1 инцидентно (инцидентна) только вершинам из V1. Иначе говоря, подграф содержит некоторые вершины исходного графа и некоторые рёбра (только те, оба конца которых входят в подграф). Подграфом, порождённым множеством вершин U называется подграф, множество вершин которого – U, содержащий те и только те рёбра, оба конца

которых входят в U. Подграф называется остовным подграфом, если множество его вершин совпадает с множеством вершин самого графа. Вершины называются смежными, если существует ребро, их соединяющее. Два ребра G1 и G2 называются смежными, если существует вершина, инцидентная одновременно G1 и G2. Каждый граф можно представить в пространстве множеством точек, соответствующих вершинам, которые соединены линиями, соответствующими ребрам (или дугам - в последнем случае направление обычно указывается стрелочками). - такое представление называется укладкой графа. Доказано, что в 3-мерном пространстве любой граф можно представить в виде укладки таким образом, что линии, соответствующие ребрам (дугам) не будут пересекаться во внутренних точках. Для 2-мерного пространства это, вообще говоря, неверно. Допускающие представление в виде укладки в 2-мерном пространстве графы называют плоскими (планарным) [11].

Другими словами, планарным называется граф, который может быть изображен на плоскости так, что его рёбра не будут пересекаться. Гранью графа, изображенного на некоторой поверхности, называется часть поверхности, ограниченная рёбрами графа.

Данное понятие грани, по - существу, совпадает с понятием грани многогранника. В качестве поверхности в этом случае выступает поверхность многогранника. Если многогранник выпуклый, его можно изобразить на плоскости, сохранив все грани. Это можно наглядно представить следующим образом: одну из граней многогранника растягиваем, а сам многогранник «расплющиваем» так, чтобы он весь поместился внутри этой грани. В результате получим плоский граф. Грань, которую мы растягивали «исчезнет», но ей будет соответствовать грань, состоящая из части плоскости, ограничивающей граф.

Таким образом, можно говорить о вершинах, рёбрах и гранях многогранника, а оперировать соответствующими понятиями для плоского графа.

Пустым называется граф без рёбер. Полным называется граф, в котором каждые две вершины смежные. Конечная последовательность необязательно различных рёбер E1,E2,...En называется маршрутом длины n, если существует последовательность V1, V2, ... Vn необязательно различных вершин, таких, что Ei = (Vi-1,Vi). Если совпадают, то маршрут замкнутый. Маршрут, в котором все рёбра попарно различны, называется цепью. Замкнутый маршрут, все рёбра которого различны, называется циклом. Если все вершины цепи или цикла различны, то такая цепь или цикл называются простыми.

Маршрут, в котором все вершины попарно различны, называется простой цепью. Цикл, в котором все вершины, кроме первой и последней, попарно различны, называется простым циклом. Граф называется связным, если для любых двух вершин существует путь, соединяющий эти вершины [12].

Любой максимальный связный подграф (то есть, не содержащийся в других связных подграфах) графа G называется компонентой связности. Несвязный граф имеет, по крайней мере, две компоненты связности. Граф называется k - связным (k - реберно - связным), если удаление не менее k вершин (ребер) приводит к потере свойства связности. Маршрут, содержащий все вершины или ребра графа и обладающий определенными свойствами, называется обходом графа. Длина маршрута (цепи, простой цепи) равна количеству ребер а порядке их прохождения. Длина кратчайшей простой цепи, соединяющей вершины vi и vj в графе G, называется расстоянием d (vi, vj) между vi и vj. Степень вершины - число ребер, которым инцидентна вершина V, обозначается D(V).

С помощью различных операций можно строить графы из более простых, переходить от графа к более простому, разбивать графы на более простые и т.д.

Среди одноместных операций наиболее употребительны: удаление и добавление ребра или вершины, стягивание ребра (отождествление пары смежных вершин), подразбиение ребра (т.е. замена ребра (u, v) на пару (u, w), (w, v), где w - новая вершина) и др.

Известны двуместные операции: соединение, сложение, различные виды умножений графов и др. Такие операции используются для анализа и синтеза графов с заданными свойствами. Два графа G1=(V1;E1), 2=(V2;E2), называются изоморфными, если существует взаимно-однозначное соответствие между множествами вершин V1 и V2 и между множествами рёбер E1 и E2, такое, чтобы сохранялось отношение инцидентности [13].

Понятие изоморфизма для графов имеет наглядное толкование. Представим рёбра графов эластичными нитями, связывающими узлы – вершины. Тогда, изоморфизм можно представить, как перемещение узлов и растяжение нитей.

Теорема 1. Пусть задан граф G=(V;E),где V - множество вершин, E - множество рёбер, тогда 2[E]=?(V), т.е. удвоенное количество рёбер равно сумме степеней вершин.

Теорема 2. (Лемма о рукопожатиях)

Теорема 3. Граф связен тогда и только тогда, когда множество его вершин нельзя разбить на два непустых подмножества так, чтобы обе граничные точки каждого ребра находились в одном и том же множестве.

Расстоянием между двумя вершинами связного графа называется длина кратчайшей цепи, связывающей эти вершины (в количестве рёбер).

Свойства связных графов.

1. Связный граф остается связным после удаления ребра тогда и только тогда, когда это ребро содержится в цикле.

2. Связный граф, имеющий К вершин, содержит по крайней мере К-1 ребро.

3. В связном графе любые две простые цепи максимальной длины имеет по крайней мере одну общую вершину.

4. В графе с N вершинами и К компонентами связности число рёбер не превышает 1/2(N-K)*(N-K+1).

5. Пусть у графа G есть N вершин. Пусть D(G)- минимальная из степеней вершин этого графа. Тогда D(G) > 1/2 (N-1).

Связный граф без циклов называется деревом.

Деревья особенно часто возникают на практике при изображении различных иерархий. Например, популярные генеалогические деревья. Пример (генеалогическое дерево): На рисунке показано библейское генеалогическое дерево [14].

Эквивалентные определения дерева.

1. Связный граф называется деревом, если он не имеет циклов.

2. Содержит N-1 ребро и не имеет циклов.

3. Связный и содержит N-1 ребро.

4. Связный и удаление одного любого ребра делает его несвязным.

5. Любая пара вершин соединяется единственной цепью.

6. Не имеет циклов и добавление одного ребра между любыми двумя вершинами приводит к появлению одного и только одного цикла.

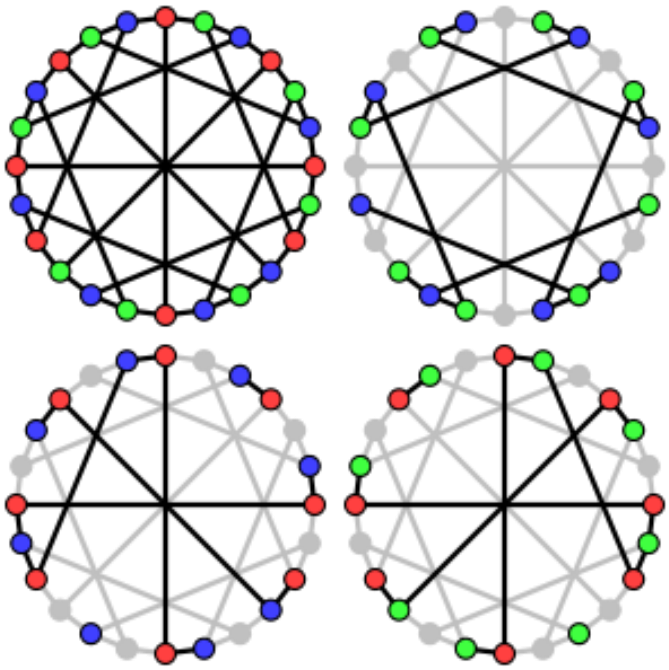

Рисунок 4 – Раскраска графов

Раскраской графа G = (V,E) называется отображение D: V(N . Раскраска называется правильной, если образы любых двух смежных вершин различны: D (U) ? D (V), если (U,V) (I. Хроматическим числом графа называется минимальное количество красок, необходимое для правильной раскраски графа (см. рисунок 4).

Теорема 5. Граф является планарным тогда и только тогда, когда он не содержит подграфа, изоморфного одному из следующих (графы Понтрягина -

Куратовского).

Свойство: В любом планарном графе существует вершина, степень которойявляется ориентированным графом, |V| = n, |E| = m. В целях упрощения изложения и избежания вырожденных случаев при оценке сложности алгоритмов будем исключать ситуации, при которых «большинство» вершин изолированные [15].

Будем также предполагать, что веса дуг запоминаются в массиве A[u, v], u, v ? V (A[u, v] содержит вес a (u, v)).

Кратчайшие пути от фиксированной вершины. Большинство известных алгоритмов нахождения расстояния между двумя фиксированными вершинами s и t опирается на действия, которые в общих чертах можно представить следующим образом: при данной матрице весов дуг A[u, v], u, v ? V, вычисляются некоторые верхние ограничения D[v] на расстояния от s до всех вершин v ?V. Каждый раз, когда мы устанавливаем, что:

D[u] + A[u, v] < D[v], оценку D[v] улучшаем: D[v] = D[u] + A[u, v].

Процесс прерывается, когда дальнейшее улучшение ни одного из ограничений невозможно.

Легко можно показать, что значение каждой из переменных D[v] равно тогда d (s, v) - расстоянию от s до v.

Заметим, что для того чтобы определить расстояние от s до t, мы вычисляем здесь расстояния от s до всех вершин графа.

Не известен ни один алгоритм нахождения расстояния между двумя фиксированными вершинами, который был бы существенным образом более эффективным, нежели известные алгоритмы определения расстояния от фиксированной вершины до всех остальных [16].

Описанная общая схема является неполной, так как она не определяет очередности, в которой выбираются вершины u и v для проверки условия минимальности расстояния. Эта очередности, как будет показано ниже, очень сильно влияет на эффективность алгоритма. Опишем теперь более детально методы нахождения расстояния от фиксированной вершины, называемой источником, его всегда будем обозначать через s, до всех остальных вершин графа.

Сначала представим алгоритм для общего случая, в котором предполагается только отсутствие контуров с отрицательной длиной. С этим алгоритмом обычно связывают имена Л.Р. Форда и Р.Е. Беллмана.

2. СПОСОБЫ ПОИСКА КРАТЧАЙШЕГО ПУТИ В ГРАФАХ

2.1 Описание элементов интерфейса программы

Рассмотрим алгоритм визуализации транспортной сети.

В функцию передаются два флага, которые позволяют временно скрывать части транспортной сети, помеченные как невидимые и как удалённые.

Текущие объекты (вершины и рёбра) изображаются с использованием полужирного начертания линий и шрифтов.

Сначала изображаются вершины, после чего строятся ребра, выходящие из этих вершин.

Каждое ребро определяется парой вершин, вершины имеют произвольные координаты. Они и определяют расположение ребра. Контуры ребра изображаются цветом графа. Внутреннее заполнение может быть выбрано индивидуально для каждого из рёбер.

На середине ребра изображается числовое значение, характеризующее вес ребра.

Предусмотрена система обозначений ребер и вершин:

× - ребро недоступно;

° - ребро помечено как невидимое;

† - ребро помечено для удаления.

После того, как отрисованы вершины, начинается визуализация ребер.

Контуры вершины имеют цвет графа, внутренняя заливка может быть установлена индивидуально для каждой вершины.

Для вершин предусмотрена та же система обозначений, что и для рёбер. Кроме этого предусмотрено ещё два возможных суффикса.

» - вершина является стартовой;

« - вершина является конечной.

При добавлении нового графа в конец списка добавляется ещё одни элемент, и он становится текущим, после чего пользователь может задать имя графа, описание, цвет и прочие характеристики.

Все вершины добавляются в левый верхний угол транспортной сети, откуда они могут быть перемещены на требуемое место. Вершины добавляются в текущий граф.

Добавление ребра осуществляется в два этапа. При нажатии на кнопку «Добавить» текущая вершина считается стартовой, и пользователь переходит в режим добавления ребра, в котором он дважды должен кликнуть на вершине конечной. Предусмотрен запрет на создание петель и дублирующийся рёбер.

Удаление элементов связано с рядом сложностей. Во-первых, для хранения объектов используется списочная структура и её реорганизация может потребовать достаточно большого времени. Поэтому выбрана стратегия, согласно которой элементы помечаются на удаление, а само удаление откладывается до прямого требования пользователя. Во-вторых, удаление элементов может потребовать удаление зависимых элементов. В связи с этим предусмотрены следующие правила:

- если граф помечен на удаление, все рёбра и вершины в него входящие так же помечаются на удаление;
- если вершина помечена на удаление, все рёбра ей инцидентные также помечаются на удаление;

Реализация этих правил осуществляется при помощи свойств, которые меняют признак пометки на удаление не только для заданного пользователем объекта, но и для всех с ним связных.

Пользователь может менять все характеристики элементов. При этом на признак доступности элемента и признак невидимости распространяются те же правила что и на признак пометки на удаление.

А все элементы управления на форме имеют соответствующие обработчики событий, которые при изменении значений сразу фиксируют их. Для хранения данных использовался формат GAF. Этот открытый формат имеет явные преимущества, т.к. является стандартом. Таким образом, транспортная сеть в этом формате потенциально может использоваться и в других приложениях [17].

Для сохранения в вышеуказанный формат использовались принципы сериализации и десериализации.

Т.к. рабочая структура данных не может быть использована для сериализации в текстовый формат (сериализации возможна только в бинарный формат данных), была разработана более простая промежуточная схема хранения данных не на основе списков, а на основе массивов.

Использовались стандартные диалоги открытия и сохранения файлов.

В этом разделе будет представлено 4 реализации алгоритма поиска оптимального маршрута, удовлетворяющего ограничениям, связанным с невозможностью превышения количества пересадок.

Все представленные алгоритмы обладают похожей секцией инициализации:

Перед осуществлением непосредственного поиска выполняются следующие действия.

1. Выполняется проверка того, что пользователь указал стартовую и конечную вершины. Эта предварительная проверка позволяет не запускать алгоритм поиска в том случае, если для его работы недостаточно исходных данных.

2. Проверяется заблокированность стартовой и конечной вершин. Эта предварительная проверка позволяет не запускать алгоритм поиска в том

случае, если в связи имеющимися ограничениями путь в принципе не может существовать.

3. Все вершины, которые ранее были помечены как часть пути, должны потерять эту отметку.

4. Все рёбра, которые ранее были помечены как часть пути, должны потерять эту отметку.

5. Создаётся стек для хранения вершин входящих в оптимальный путь. Создаётся стек для хранения рёбер входящих в оптимальный путь. По идее этот стек является избыточным, т.к. рёбра могут быть восстановлены из последовательности вершин, но его наличие позволяет получить к рёбрам непосредственный доступ без дополнительных затрат на их поиск. Использование стека для хранения обусловлено тем фактом, что для поиска в глубину эта структура данных является более удобной и естественной.

6. Начальная стоимость оптимального пути принимается за максимальное целое число. Подобное допущение возможно в силу того, что транспортная сеть является конечной и в качестве весовых коэффициентов рёбер используются целые положительные числа.

7. Создаётся стек для хранения вершин, входящих в текущий путь. Создаётся стек для хранения рёбер, входящих в текущий путь. Причины, по которым для хранения используется стек аналогичны перечисленным в пункте 5.

8. Вес текущего пути принимается за ноль.

9. Количество сделанных пересадок принимается за ноль. После осуществления поиска оптимального маршрута выполняются действия необходимые для его визуализации.

10. Делается проверка того, изменила ли значение переменная, в которой должен храниться вес оптимального пути. Перед запуском алгоритма поиска маршрута эта переменная имела значение равное максимальному целому числу. Если переменная не изменила значение,

делается вывод о том, что при заданных условиях ни один путь между начальной и конечной вершинами не может быть найден, и пользователю выводится соответствующее сообщение.

11. Из стека извлекаются все рёбра, вошедшие в оптимальный маршрут. Каждое ребро, извлечённое из стека, помечается как часть оптимального маршрута. Для каждого ребра, извлечённого из стека, определяются образующие его вершины и так же помечаются как часть оптимального маршрута.

12. Осуществляется отрисовка транспортной сети с отмеченным на ней маршрутом [18].

Перейдём к самому алгоритму поиска. Данная версия алгоритма поиска в глубину была реализована с использованием рекурсии, т.к. рекурсия является очень естественным приёмом при работе с графами. Учитывая тот факт, что использование рекурсии может приводить к замедлению работы алгоритма в следующем параграфе будет рассмотрена версия алгоритма поиска в глубину без использования рекурсии.

Основная идея алгоритма состоит в следующем.

В функцию передаётся 4 параметра:

1. Вершина, в которой мы сейчас находимся (из какой вершины пришли).
2. Вершина, в которую мы хотим пойти.
3. Ребро, по которому мы хотим пойти.
4. Вершина, в которую мы хотим попасть.

Функция содержит избыточные данные (например, зная текущую и следующую вершину мы легко можем восстановить ребро, по которому мы хотим пойти), но это упрощает реализацию алгоритма, т.к. не требует организации дополнительного поиска в графовой структуре данных.

Стеки вершин и рёбер, а также стоимости текущего и оптимального пути задаются при помощи глобальных переменных.

При первом запуске алгоритма, мы не находимся ни в одной из вершин, поэтому первый параметр функции принимает значение минус единица. Аналогичная ситуация складывается с третьим параметром. В связи с тем, что мы попадаем в стартовую вершину не через какое-либо из существующих рёбер, то этот параметр так же равняется минус единице. Второй параметр соответствует стартовой вершине. Четвёртый параметр соответствует той вершине, в которую мы хотим попасть.

Перейдём к штатному режиму работы алгоритма.

В-первую очередь проверяется «А можно ли идти в эту вершину?».

Идти нельзя если:

- мы уже были в этой вершине (если пренебречь этим условием, то возможно образование циклов, а т.к. веса рёбер у нас всегда положительные, то путь, в котором есть цикл, никогда не сможет стать оптимальным);
- вершина заблокирована;
- ребро заблокировано.

Если нельзя, то функция досрочно прекращает свою работу.

Во-вторых, проверяется «А нужно ли идти в эту вершину?».

Идти не нужно если:

- если мы получим путь более тяжёлый, чем текущий оптимальный (эта эвристика позволяет нам не рассматривать полностью все пути и сокращать перебор, отсекая ненужные ветви, в том случае, если заведомо известно, что маршрут с текущим префиксом не сможет быть оптимальным);
- если мы получим количество пересадок, большее, чем заложено в ограничении (это основное отличие классического алгоритма поиска в глубину от рассматриваемого при решении поставленной задачи) [20].

Если не нужно, то функция досрочно прекращает свою работу.

После этих проверок вершина, которая подходит по всем параметрам, добавляется в стек вершин, в котором хранится текущий путь.

Затем пересчитываются показатели.

- увеличивается вес текущего пути (соответственно на вес ребра, по которому был сделан очередной шаг алгоритма поиска в глубину);
- при необходимости увеличивается количество пересадок.

Делается проверка «А не достигли ли мы заданной вершины?».

Если вершина достигнута, то делается проверка на то, а не является ли текущий путь более оптимальным (в смысле суммарного веса), по сравнению с текущим кандидатом на оптимальность. Если текущий путь лучше, то он записывается на место оптимального [21].

Далее осуществляется шаг рекурсии. Для всех рёбер, выходящих из текущей вершины, делается рекурсивный вызов, тем самым обеспечивается полный перебор, который и требуется для решения поставленной задачи.

При нормальном завершении работы функции необходима реализация корректного возврата из рекурсии, т.е. должны быть пересчитаны показатели:

- уменьшается вес текущего пути (т.к. мы движемся обратно по маршруту);
- при необходимости уменьшается количество пересадок.

Поиск в глубину (нерекурсивная версия).

Нерекурсивная версия этого алгоритма может иметь выигрыш по скорости работы в связи с тем, что:

- каждый вызов рекурсивной функции приводит к тому, что происходит надстраивание программного стека, сохраняется точка возврата, создаётся окружение для запуска функции, выделяется память под локальные переменные и т.д.
- увеличиваются требования по памяти;
- увеличиваются требования по количеству выполняемых инструкций.

Кроме того, алгоритм уже имеет в своём составе стек, в котором хранится текущий путь. В принципе, при использовании рекурсивной версии поиска в глубину есть возможность избежать необходимости хранения

пройденного пути в стеке, т.к. этот путь может быть восстановлен при обратном ходе рекурсии. Кроме того, современные процессоры и компиляторы имеют ряд механизмов, оптимизирующих рекурсивные вызовы таким образом, что накладные расходы будут практически не заметны. Но, учитывая, что в данном случае решается не классическая задача поиска в глубину, этот подход может вызвать ряд проблем.

В нерекурсивную функцию поиска в глубину передаётся только два параметра: стартовая вершина, и вершина, в которую мы хотим попасть.

Рассмотрим сам алгоритм.

1. Осуществляется проверка того, не является ли стартовая вершина совпадающей с финальной вершиной. Если это так, то осуществляется досрочный выход из функции поиска маршрута. При этом ни одно ребро не может быть помечено как часть оптимального пути т.к. согласно принятым выше ограничениям в графе не могут присутствовать петли. Поэтому только одна вершина добавляется в список вершин, входящих в оптимальный путь.

2. Для каждой вершины заводится вспомогательный массив, в котором хранится порядковый номер ребра, которое было выбрано перед переходом в следующую вершину. В рекурсивной версии алгоритма не было необходимости в этой переменной, т.к. эта информация сохранялась перед рекурсивным вызовом в локальной переменной. Элементы этого вспомогательного массива инициализируются значением минус единица, т.к. ни одно ребро ещё не было выбрано.

3. В стек складывается текущая вершина, с которой начинается поиск.

4. Поиск продолжается до тех пор, пока не будут просмотрены все возможные маршруты. Учитывая специфику алгоритма поиска (которая будет рассмотрена ниже) при возврате назад, все посещённые ранее вершины постепенно извлекаются из стека. Поэтому условием завершения поиска является извлечение из стека последней вершины (т.е. когда он становится

пустым). В рекурсивной версии мы имеем почти аналогичное условие, только там пустым становится стек рекурсивных вызовов.

Верхняя вершина стека является текущей. Осуществляется проверка того, а не дошли ли мы до финальной вершины. Если мы дошли до финальной вершины, то необходимо выполнить уже стандартную проверку того, а не является ли найденный путь более лёгким по сравнению с текущим кандидатом на оптимальный маршрут. Если это действительно так, то текущий путь запоминается как оптимальный и его вес так же запоминается. Далее нам необходимо сделать шаг назад, что бы мы могли проверить альтернативные пути, исходящие из предыдущей вершины. Для этого нам необходимо просмотреть ребро, находящееся на вершине стека рёбер, и узнать из какой вершины мы пришли. Смысла дальнейшего передвижения из финальной вершины по графу нет, поэтому далее делается шаг назад: из стека вершин текущего маршрута извлекается верхняя вершина; из стека рёбер текущего маршрута извлекается верхнее ребро; вес текущего маршрута уменьшается на вес извлечённого ребра; если возврат назад на один шаг связан с пересадкой, то текущее количество сделанных пересадок уменьшается на единицу. В том случае, если мы ещё не дошли до финальной вершины нам необходимо просмотреть все рёбра, по которым может быть осуществлено движение из текущей вершины. При этом отсекаются пути, которые приводят нас в вершины, которые уже хранятся в текущем пути. Это делается в силу того, что маршрут, содержащий цикл никогда не может претендовать на статус оптимального в плане суммарного веса посещённых рёбер. Так же не рассматриваются заблокированные рёбра и рёбра, которые приводят в заблокированные вершины. В силу ограничений на максимальное количество пересадок не рассматриваются рёбра, которые приводят к превышению заданного количества. Для первого ребра, прошедшего вышеуказанную проверку, выполняется следующий набор действий. вершина, в которую попадает ребро, добавляется в стек вершин и становится

текущей. ребро добавляется в стек, в котором хранится текущий маршрут на этом цикл завершается, т.к. делается шаг в следующую вершину, но для вершины, из которой делается шаг, запоминается на каком ребре была сделана остановка, с тем, чтобы по возвращению в эту вершину, продолжить перебор возможных маршрутов. После того, как все рёбра, выходящие из текущей вершины, проверены, делается шаг назад. Для этого из стека вершин текущего маршрута извлекается верхняя вершина, из стека рёбер текущего маршрута извлекается верхнее ребро; вес текущего маршрута уменьшается на вес извлечённого ребра, если возврат назад на один шаг связан с пересадкой, то текущее количество сделанных пересадок уменьшается на единицу и так как мы ушли из текущей вершины, счётчик просмотренных из неё рёбер обнуляется, т.е. принимает значение минус единица [22].

Очевидно, что этот алгоритм является самым эффективным в плане использования оперативной памяти, т.к. мы храним только один текущий путь.

Рассмотрим вопрос, связанный со скоростью работы. Все представленные в этой работе алгоритмы основаны на полном переборе. Во всех алгоритмах используются одинаковые эвристики, позволяющие отсекать заведомо неоптимальные ветви. Следовательно, с этой точки зрения ускорение возможно по следующим направлениям:

- отказаться от рекурсии (что и было сделано в рамках данного алгоритма);
- использование избыточных данных, позволяющих хранить уже рассчитанные значения (что и было сделано во всех предложенных алгоритмах);
- использование многопоточных алгоритмов, рассчитанных на выполнение на многоядерных процессорах (что и будет сделано в следующих разделах).

2.2 Алгоритмы поиска в ширину

Поиск в ширину (однопоточная версия).

Функция поиска в ширину является более требовательной в плане использования оперативной памяти, т.к. ней все пути рассматриваются параллельно. Особенно эта проблема становится актуальной в графах с большим количеством связей. Поэтому однопоточная версия этого алгоритма не желательна для использования и приводится в силу того, что на её основе будет строиться многопоточная версия.

Алгоритму на вход подаётся два параметра, номер стартовой вершины и номер конечной вершины.

Перед запуском алгоритма осуществляется проверка на то, не совпадают ли стартовая и конечная вершина. В случае совпадения производится досрочный выход из функции подобно тому, как это делалось в предыдущем алгоритме.

Как и раньше текущий путь будет размещаться в структуре данных типа стек. Туда помещается стартовая вершина.

В классической версии поиска в ширину используется структура данных типа очередь. Однако мы не можем использовать только очередь, т.к. нам важен не только обход графа в заданном порядке, нам необходимо иметь возможность хранения всего пройденного пути и сравнения его с оптимальным. Именно в силу этого условия будет использована не просто очередь, а очередь стеков (в каждом стеке будет храниться один из потенциальных путей) [23].

На самом деле будет использовать 4 очереди.

В первой очереди будут храниться стеки вершин потенциальных маршрутов.

Во второй очереди будут храниться стеки рёбер потенциальных маршрутов.

В третьей очереди будут храниться веса потенциальных маршрутов.

И в четвёртой очереди будут храниться количества пересадок, сделанных на текущий момент для каждого из потенциальных маршрутов.

По мере достижения финальной вершины, маршруты будут удаляться из очередей. Поэтому условием окончания работы алгоритма будет тот факт, что одна из очередей будет пуста. Пусть это будет очередь вершин.

Основной цикл до опустения очереди вершин будет выглядеть следующим образом:

1. Из начала очереди маршрутов из вершин извлекается маршрут и делается текущим.

2. Из начала очереди маршрутов из рёбер извлекается маршрут и делается текущим.

3. Из начала очереди весов извлекается вес текущего маршрута.

4. Из начала очереди количества пересадок извлекается текущее количество пересадок.

5. Извлекаем с вершины стека вершин текущего маршрута вершину и делаем её текущей.

6. Если текущая вершина совпадает с финальной вершиной, то осуществляется проверка того, а не является ли текущий путь более лёгким, по сравнению с текущим, считающимся оптимальным маршрутом. И если это так, то он записывается на его место и его вес также запоминается.

7. Если же текущая вершина не является финальной, то нам необходимо сформировать все возможные направления движения из неё исходящие. Для этого просматриваются все рёбра, которые выходят из текущей вершины. Не рассматриваются рёбра, которые приводят к возникновению циклов, т.е. приводят к вершинам уже находящимся в текущем пути. Так же не рассматриваются заблокированные рёбра и рёбра,

приводящие в заблокированные вершины. Не рассматриваются рёбра, имеющие такой вес, которые будучи сложенным с текущим весом, будет превышать вес текущего оптимального маршрута. Не рассматриваются рёбра, которые ведут к превышению максимально возможного количества пересадок. После того, как подходящие ребра найдены, для каждого из них осуществляется следующая последовательность действий: в стек вершин добавляется конечная вершина ребра; в стек рёбер добавляется ребро; пересчитывается текущий вес за счёт прибавления веса ребра, по которому осуществляется переход; пересчитывается текущее количество пересадок, в соответствии с ребром по которому осуществляется переход; копия стека вершин помещается в конец очереди стеков вершин;. копия стека рёбер помещается в конец очереди стеков рёбер; вес текущего пути помещается в конец очереди весов; количество пересадок текущего пути помещается в конец очереди количества пересадок; делается шаг назад путём извлечения вершины из стека текущих вершин и извлечением ребра из стека текущих рёбер, также пересчитывается текущий вес путём вычитания веса удалённого ребра и пересчитывается количество сделанных пересадок в сторону уменьшения [25].

Поиск в ширину (многопоточная версия)

Этот метод является улучшенной версией поиска в ширину. Он сохраняет тот же самый принцип обхода графа, но позволяет одновременно просматривать сразу несколько потенциальных маршрутов. Распараллеливание поиска в глубину невозможно в силу специфики порядка обхода вершин. В данной версии алгоритма не осуществляется ограничение на максимальное количество одновременно работающих потоков, и при необходимости, может быть осуществлено за счёт использования механизма пула потоков.

В силу многопоточности для запуска алгоритма используются дополнительные построения.

1. Заводится глобальная переменная, в которой хранится текущее количество активных потоков. Каждый поток при запуске увеличивает её значение и уменьшает перед завершением.

2. Заводится вспомогательная структура данных, хранящая в себе полный набор параметров необходимых для алгоритма. Это делается в силу того, что функции-делегату представляющей собой тело потока может быть передан только один параметр типа object. В состав структуры входят следующие поля: текущая вершина; конечная вершина; текущий вес; текущее количество пересадок; стек вершин, входящих в текущий путь; стек рёбер, входящих в текущий путь.

После запуска потока для осуществления поиска в ширину главный поток, отвечающий за работу формы должен дождаться завершения всех вспомогательных потоков. Для этого он ждёт завершения первого созданного потока, а затем ждёт пока переменная, в которой хранится текущее количество активных потоков, не станет равной нулю. Отдельное ожидание завершения первого порождённого потока необходимо в силу того, что он может не успеть инициализироваться и инкрементировать значение переменной, в которой хранится текущее количество активных потоков.

Рассмотрим сам алгоритм:

1. Увеличивается количество активных потоков.
2. В локальный стек вершин помещается текущая вершина.
3. Если текущая вершина совпадает с финальной вершиной, то необходимо проверить, не является ли текущий путь лучшим по сравнению с кандидатом на оптимальность. Однако, подобная проверка может быть осуществлена только в рамках критической секции, которая помогает избежать потенциальных конфликтов с параллельными потоками. И, если текущий путь лучше оно копируется в качестве оптимального и его вес запоминается (эти два действия так же должны осуществляться в рамках критической секции).

4. Если же мы ещё не дошли до финальной вершины. Перебираются все рёбра исходящие из текущей вершины. Не рассматриваются рёбра, которые приводят к возникновению циклов, т.е. приводят к вершинам уже находящимся в текущем пути. Так же не рассматриваются заблокированные рёбра и рёбра, приводящие в заблокированные вершины. Не рассматриваются рёбра, имеющие такой вес, которые будучи сложенным с текущим весом, будет превышать вес текущего оптимального маршрута. Не рассматриваются рёбра, которые ведут к превышению максимально возможного количества пересадок. После того, как подходящие ребра найдены, для каждого из них осуществляется следующая последовательность действий: формируется копия текущего стека вершин;. формируется копия текущего стека рёбер; формируется копия веса текущего пути;. формируется копия количества пересадок на текущем пути; в копию стека вершин помещается конец ребра, по которому мы хотим идти; в копию сетка рёбер помещается ребро, по которому мы хотим идти; копия веса текущего пути увеличивается на вес ребра, по которому мы хотим пойти; копия количества пересадок на текущем пути, если ребро ведёт в другую транспортную сеть; и вместо того чтобы помещать это всё в конец очереди, как это делалось в классическом поиске в ширину создаётся ещё один поток с только что сформированным набором параметров.

5. После того как все возможные пути из текущей вершины перебраны и соответствующие потоки стартовали, поток уничтожается, уменьшая количество запущенных потоков на единицу.

Пример работы алгоритма

Рассмотрим небольшой пример изображенный на рисунке (см. рисунок 5). Пусть необходимо найти путь из вершины 1 в вершину 9.

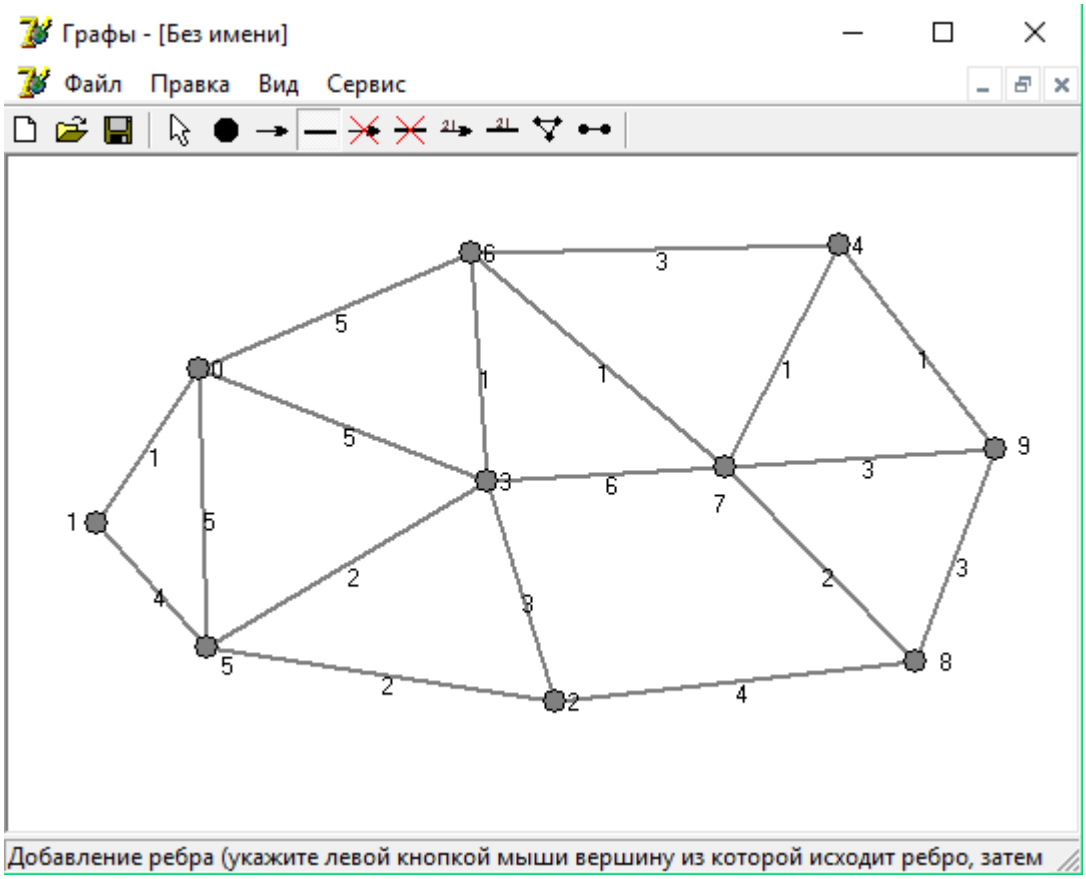

Рисунок 5 – Дерево графа

Кратчайший путь будет найден на основе тех алгоритмов, которые были изложены выше. Результат которые будет выведен демонстрируется на рисунке (см. рисунок 6).

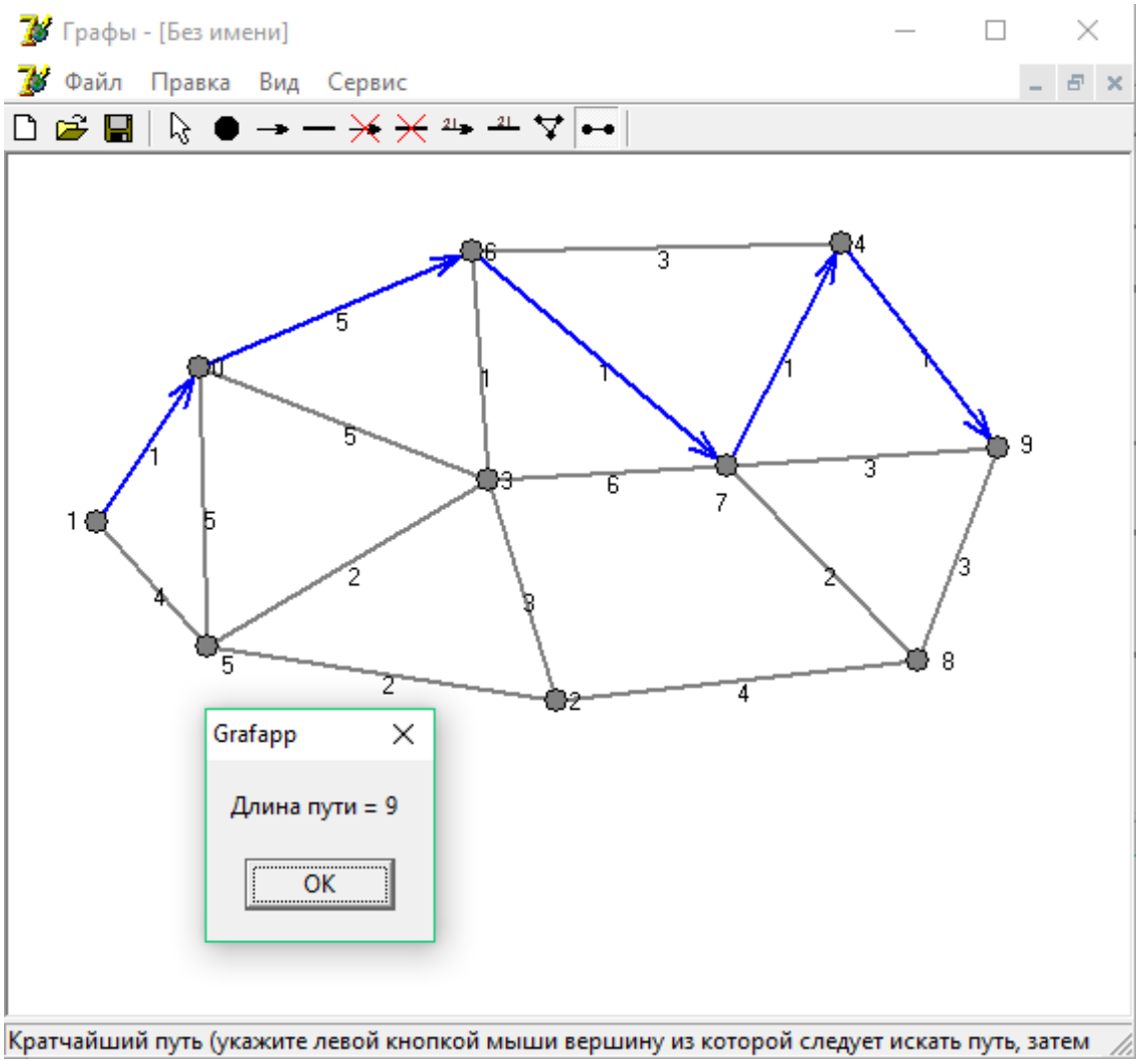

Рисунок 6 – Нахождение минимального пути

Начав перебирать вершины, алгоритм сравнивает варианты выбора вершины, в которую будет совершен переход, веса расстояний между точками, как раз это наглядно демонстрируют, в конечном итоге будет выведен самый кратчайший маршрут и его длина.

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ГОТОВОГО ПРОДУКТА

3.1 Использование теории графов в учебной программе

Проблема обучения школьников поиску решения математических задач, является одной из наиболее трудных в теории и методике обучения математике. Проявляется она прежде всего в том, что математически способные учащиеся не могут найти способ решения, зная весь необходимый для этого теоретический материал.

Актуальным для каждого педагога является вопрос о том, какие средства использовать при обучении. Средства обучения стали не только источником учебной информации, но и инструментом управления познавательной деятельностью школьников. Средства обучения должны содействовать усвоению основ наук, развитию мышления, формированию мировоззрения, воспитанию учащихся в духе нравственности. Они должны быть приспособлены к труду учителя и учащихся, обеспечивать применение эффективных методов и приемов работы [26].

Решая задачи при изучении элементов теории графов, необходимо помнить, что в каждом шаге, в каждом этапе ее решения необходимо применить творчество. С самого начала, на первом этапе, оно заключается в том, суметь проанализировать и закодировать условия задачи. Второй этап - схематическая запись, состоит в геометрическом представлении графов, и на этом этапе элемент творчества очень важен потому, что далеко не просто найти соответствия между элементами условия и соответствующими элементами графа. Все остальные этапы тоже не обходятся без применения творчества и изобретательности. Проведение поиска способа и осуществления решения задачи (с проверкой и исследованием) нуждается в следующих способностях решающих: способность абстрагирования, способность моделирования, способность гибкого применения теории

графов, способность применения всех известных математических способов решения. Бесспорно, формулирование ответа задачи — это тоже творческое изобретение, т.к. также необходима и кодировка, и абстрагирование.

Облегчение восприятия и усвоения учащимися математических знаний может быть достигнуто разумным использованием различных средств наглядности - таблиц, чертежей и рисунков и т. д.

Рассмотрим некоторые средства наглядности для обучения поиску решения задач по теме Элементы теории графов.

Задача 1. В деревне 9 домов. Известно, что у Петра соседи Иван и Антон, Максим сосед Ивану и Сергею, Виктор - Диме и Никите, Евгений - сосед Никиты, а больше соседей в этой деревне нет (соседними считаются дворы, у которых есть общий участок забора). Может ли Пётр огородами пробраться к Никите за яблоками?

Решением данной задачи будет служить граф, где точками будут обозначены дома, соединенные между собой линиями только те из них, которые являются соседними. Это наглядно демонстрирует, что пробраться огородами из дома Петра к дому Никиты нельзя [30].

Данная задача наглядно демонстрирует, что для быстрого и простого решения задачи, можно использовать граф, на котором будут изображены допустимые решения или указан минимальный путь от точки А к точке В.

3.2 Реализация программного кода

Данная программа имеет 3 формы: MainForm, SmegGrafForm, SetarcWeightForm.

Форма MainForm отвечает за главный интерфейс программы, где располагаются элементы управления и осуществляется ввод и вывод

информации. Так же здесь содержаться процедуры, описанные в других формах. Код данной формы представлен в приложении А.

SmegGrafForm отвечает за представление графа в виде матрицы смежности. При создание нового графа можно задать его в виде матрицы, указав вершины и расстояние между ними. Код данной формы представлен в приложении Б.

Форма SetarcWeightFormUnit отвечает за вес расстояния между вершинами графа. Данные после построения вершин, когда пользователь соединяет вершины направленными или ненаправленными маршрутами. Код данной формы представлен в приложении В [41].

Данные в программу вводятся 3 способами.

Первый способ ввода данных осуществляется при помощи файла (см. рисунок 7), второй при помощи заполнения матрицы смежности внутри самой программы (см. рисунок 8) и третий это произвольная расстановка вершин и их соединение с указанием веса пути (см. рисунок 9).

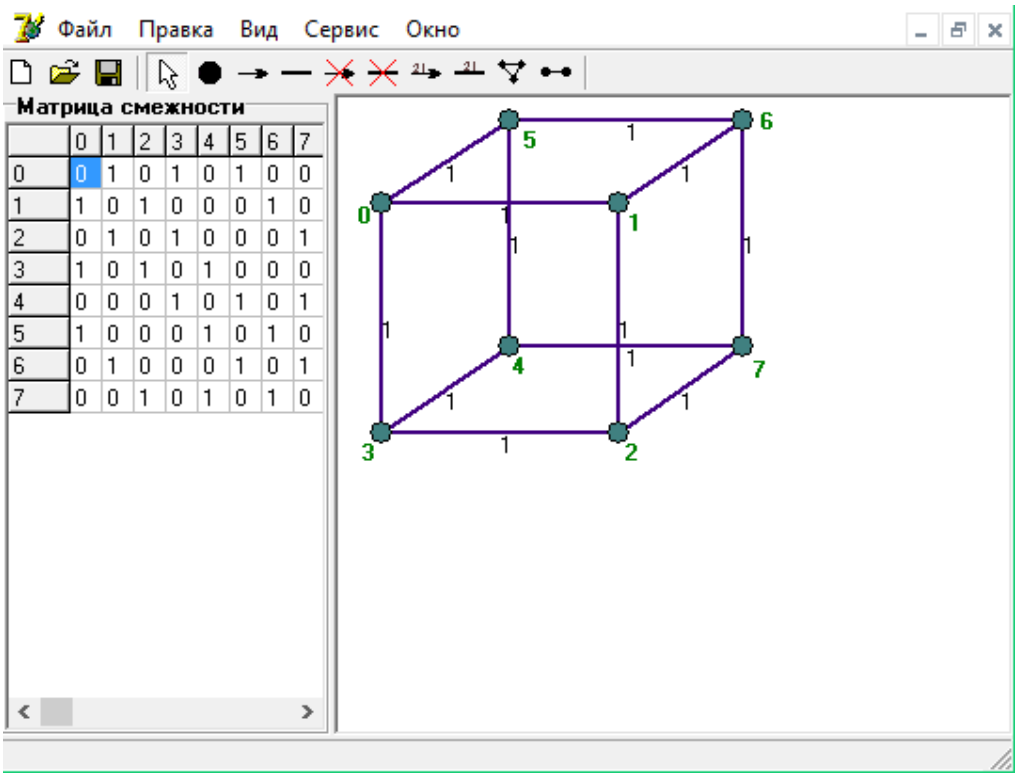

Рисунок 7 – Файловый ввод данных

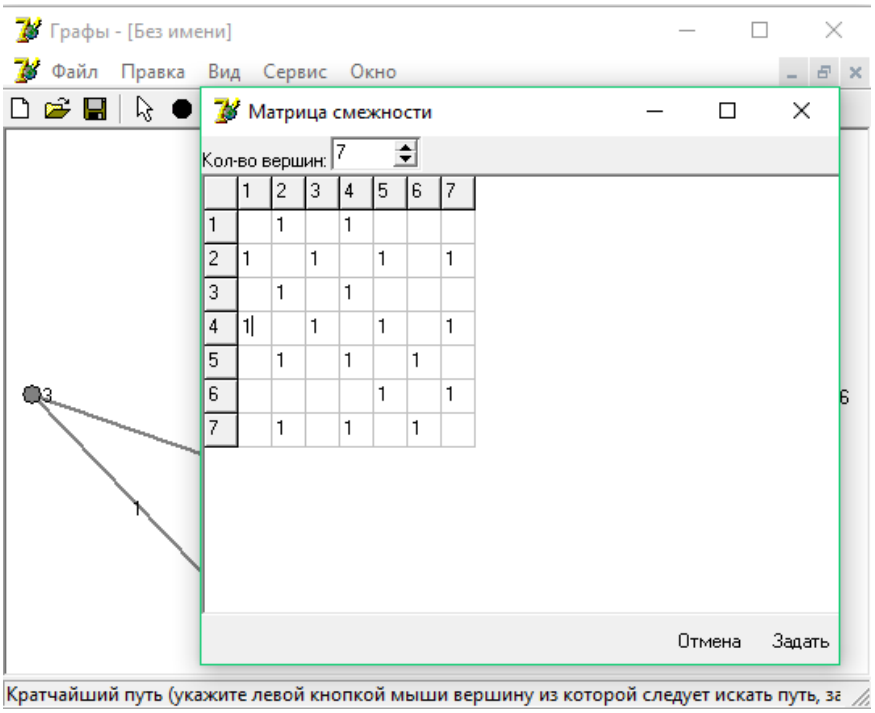

Рисунок 8 – Матрица смежности

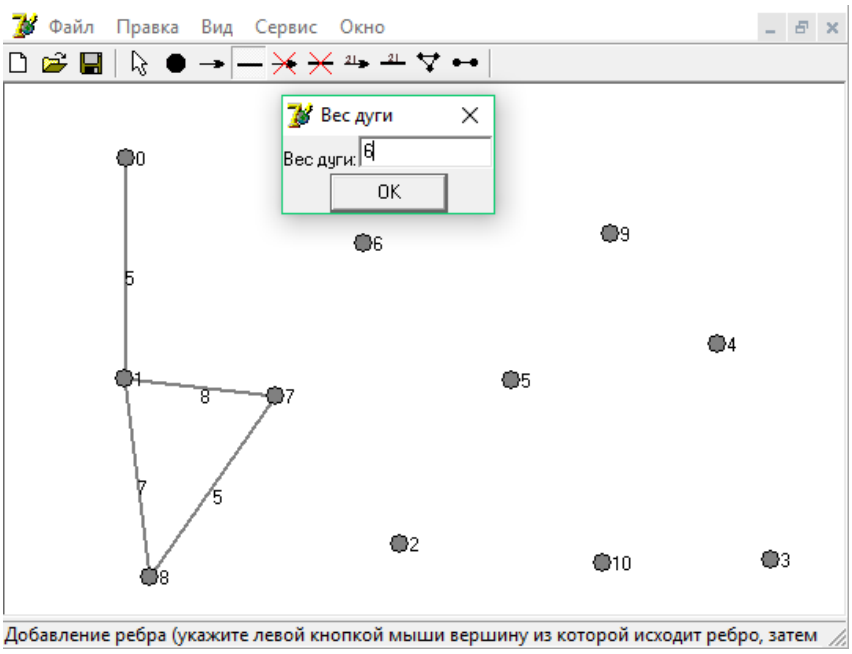

Рисунок 9 – Произвольная расстановка вершин

Имея различные способы ввода, можно сказать что программа будет удобна в любой ситуации и в зависимости от данных которые представлены,

их можно будет загрузить в программу. Так же, при необходимости, конечный результат можно будет сохранить, чтобы передать его другому пользователю.

ЗАКЛЮЧЕНИЕ

Теория графов в настоящее время является интенсивно развивающимся разделом дискретной математики. Это объясняется тем, что в виде графовых моделей описываются многие объекты и ситуации: коммуникационные сети, схемы электрических и электронных приборов, химических молекул, отношение между людьми и многое другое.

Т.е. учащиеся, добыв первоначальные знания с помощью занимательных задач, переходят к закреплению и развитию этих знаний на базе решения более сложных задач.

Теория графов привлекательна еще и тем, что в ней наряду с решенными задачами и проблемами существуют задачи нерешенные.

А это является малой долей изученного в данной теории и до сих пор остается мощным стимулом для дальнейших исследований различных свойств графов.

Современная теория графов находит ряд интересных и важных приложений в других разделах математики, физики, в теории жидких кристаллов, в молекулярной биологии, кибернетике, вычислительной технике и т.д.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Белкин, А.Р., Шумов, С.И. Анализ и оценка традиционных и нетрадиционных механизмов получения и обобщения новых знаний. Региональная программа / А.Р. Белкин, С.И. Шумов // Информатика и образование. – 2014. – № 6.

2. Босова, Л.Л. Макроязык оператора графического вывода DRAW / Л.Л. Босова // Информатика и образование. – 2015. – № 5.

3. Бурцева, Г. А. Графика в обучении программированию / Г.А. Бурцева // Информатика и образование. – 2012. – № 6.

4. Воронцова, Ю.Л. Знакомство с графикой в Бейсике / Ю.Л. Воронцова // Информатика и образование. – 2011. – № 6.

5. Винницкий, Ю.А. Принципы разработки мультимедийных учебников для средней школы / Ю.А. Винницкий, Г.М. Нурмухамедов // Информатика и образование. – 2016. – № 10. – С. 125.

6. Вуль, В. Электронные издания: учебное пособие / В. Вуль. – М., 2013. – 535 с.

7. Вымятнин, В.М. Дистанционное образование и его технологии / В.П. Демкин, В.Ф. Нявро. – Томск, 2012. – 376 с.

8. Демкин, В.П. Классификация образовательных электронных изданий: основные принципы и критерии / В.П. Демкин, Г.В. Можаева // Открытое и дистанционное образование. – 2013. – № 11-12. – С. 3-6.

9. Дрепа, Е.Н. Положение о выпускной квалификационной работе (дипломной работе) / Е.Н. Дрепа, С.Р. Новикова. – Нижнекамск: Изд-во НМИ, 2016. – 40 с.

10. Епанешников, А.М., Епанешников В.А. Программирование в среде TURBO-PASCAL 7.О.: учебное пособие / А.М. Епанешников, В.А. Епашнешников. –М.: "ДИАЛОГ-МИФИ", 2015. – 282 С.

11. Журбина, Н.А. Информационно-коммуникационные технологии в образовании / Н.А. Журбина // Информационное общество. – 2011. – № 2. – С. 5-6.

12. Зайнутдинова, Л.Х. Создание и применение электронных учебников (на примере общетехнических дисциплин) / Л.Х. Зайнутдинова. – Астрахань: ЦНТЭП, 2014. – 206 с.

13. Зельднер, Г.А. Программируем на языке QuickBasic 4.5: учебное пособие по курсам «Информатика и вычислительная техника», «Основы программирования» / Г.А. Зельднер. – М.: ABF, 2016. – 432 с.

14. Интернет-обучение: технологии педагогического дизайна / М.В. Моисеева, Е.С. Полат, М.Ю. Бухаркина, М.И. Нежурина. – М.: Камерон, 2014. – 216 с.

15. Информационные технологии в образовании и науке: научно-технический отчет. – Томск, 2013.

16. Казаков, В.Г., Дорошквин А.А. Лекционная мультимедиа аудитория / В.Г. Казаков, А.А. Дорошквин // Информатика и образование. – 2015. – № 4.

17. Коджаспирова, Г.М. Технические средства обучения и методика их использования: уч. пособие для студентов высших учебных заведений / Г.М. Коджаспирова, К.В. Петров. – М.: Академия, 2011. – 304 с.

18. Кривошеев, А.О. Разработка и использование компьютерных обучающих программ: учебник / А.О. Кривошеев // Информационные технологии. – 2011. – № 2. – С. 14-17.

19. Кудинова, В.И. О пользе программирования для школьников / В.И. Кудинова // Информатика и образование. – 2012. – № 11.

20. Лапчик, М.П. Методика преподавания информатики: учебник / М.П. Лапчик. – М.: Академия, 2013. – 624 с.

21. Лебедева, М. Б. Принципы построения и методика применения электронного учебно-методического комплекса / М.Б. Лебедева //

Информационные и коммуникационные технологии в образовании. – СПб. : Изд-во БАН, 2015. –472 с.

22. Лобачев, С.Л. Универсальная инструментальная информационно-образовательная среда системы открытого образования Российской Федерации: лекция-доклад / С.Л. Лобачев, А.А. Поляков // Информационные технологии в управлении качеством образования и развитии образовательного пространства. – М.: Исследовательский центр проблем качества подготовки специалистов, 2011. – 40 с.

23. Марченко, А.И. Марченко Л.А. Программирование в среде Turbo Pascal 7.0: учебное пособие / А.И. Марченко, Л.А. Марченко. – Киев: "Век+". – 2010. – 460с.

24. Методы, и средства разработки электронных изданий // http:// www.mi.ru/~dupliksv/pauk/soder.html.

25. Можаева, Г.В. Как подготовить мультимедиа курс: методическое пособие для преподавателей / Г.В. Можаева, И.В. Тубалова. – Томск: Изд-во Том.ун-та, 2012. – 264 с.

26. Можаева, Г.В. Электронные ресурсы в историческом образовании / Г.В. Можаева, А.В. Фещенко // Открытое и дистанционное образование, 2014. – № 2 (14). – С. 13-21.

27. Начинская, С.В. Спортивная метрология: учеб. пособие для студ. высш. учеб. заведений. – М.: Издательский центр «Академия», 2015. – 240 с.

28. Новые педагогические и информационные технологии в системе образования / под ред. Е.С. Полат. – М.: Академия, 2001. – 213 с.

29. Осин, А.В. Предпосылки концепции образовательных электронных изданий / А.В. Осин // Основные направления развития электронных образовательных изданий и ресурсов: материалы научно-практической конференции. – М.: Республиканский мультимедиа центр, 2012. – 167 с.

30. Окулов, С.М. Основы программирования: 3-е изд. / С.М. Окулов – М. : БИНОМ. Лаборатория знаний, 2016. – 310 с.

31. Открытое образование – объективная парадигма XXI века / Ж.Н. Зайцева, Ю.Б. Рубин, Л.Г. Титарев, В.П. Тихомиров, А.В. Хорошилов, В.Л. Усков. – М.: МЭСИ, 2010. – 25 с.

32. Буланова М.В., Духавнева А.В, Столяренко Л.Д. [и др.]. Педагогика и психология высшей школы / – Ростов н/Д.: Феникс, 2013. – 314 с.

33. Петров, В.И. Графические средства алгоритмического языка TURBO-PASCAL: методические указания к выполнению лабораторных работ / В.И. Петров. – СПб., 2012. – 33 с.

34. Программирование на языке Паскаль. Задачник / под ред. О.Ф. Усковой. – Спб. : "Питер". – 2002. – 334с.

35. Попов В.Б. Turbo Pascal для школьников: учебное пособие / В.Б. Попов. – М.: Финансы и статистика, 2012. – 234 с.

36. Семакин, И. Информатика и ИКТ. Базовый курс: учебник для 9 класса. – 2е изд. / И. Семакин. – М.: Бином. Лаборатория знаний, 2016. – 359 с.

37. Семакин, И. Г., Шестаков А. П.Основы программирования: учебное пособие / И.Г. Семакин, А.П. Шестаков. – М.: Лаборатория Базовых Знаний, 2013. – 312 с.

38. Симонович, С. Специальная информатика: универсальный курс / С. Симонович, Г. Евсеев, А. Алексеев. – М.: АСТ-ПРЕСС, Инфорком-Пресс, 2012. – 480 с.

39. Субботин, М.М. Новая информационная технология: создание и обработка гипертекстов. – М., 2012. – 174 с.

40. Немнюгин С.А. TurboPascal: учебное пособие / под ред., – Спб.: Питер, – 2013. – 496 с.

41. Минько Э.В., Покровского А.В. Технико-экономическое обоснование исследовательских и инженерных решений в дипломных проектах и работах: учебное пособие / под ред. – Свердловск Уральский университет, 2010. – 144 с.

42. Уваров, А.Ю. Электронный учебник: теория и практика / А.Ю. Уваров. – М.: УРАО, 2014. – 220 с.

43. Угринович, Н.Д. Информатика и информационные технологии: учебник для 10-11классов / Н. Д. Угринович. – М.: БИНОМ. Лаборатория знаний, 2015. – 511 с.

44. Угринович, Н.Д. Практикум по информатике и информационным технологиям: учебное пособие для образовательных учреждений / Н.Д. Угринович, Л.Л. Босова, Н.И. Михайлова. – М.: БИНОМ. Лаборатория знаний, 2014. – 394 с.

45. Угринович, Н.Д. Информатика. Базовый курс: учебник для 9 класса / Н.Д. Угринович – М.: БИНОМ. Лаборатория знаний, 2016. – 304 с.

46. Угринович, Н.Д. Преподавание курса «Информатика и ИКТ» в основной и старшей школе: методическое пособие / Н.Д. Угринович. – М.: БИНОМ. Лаборатория знаний, 2014. – 139 с.

47. Фаронов, В.В. Турбо Паскаль: в 3 т. / В.В. Фаронов. – М.: Учебно-инженерный центр "МВТУ-ФЕСТО ДИДАКТИК", 2012. – 286 с.

48. Федоренко, Ю. Алгоритмы и программы на Qbasic: учебный курс / Ю. Федоренко. – СПб.: Питер, 2012. – 288с.

49. Хомоненко, А.Д. Основы современных компьютерных технологий / А.Д. Хомоненко. – СПб.: КОРОНА, 2013. – 448 с.

50. Чернов, Б.И. Программирование на алгоритмических языках Бейсик, Фортран, Паскаль: учебное пособие / Б.И. Чернов. – М.: Просвещение, 2011. – 192 с.

51. Шикин, Е.В. Начала компьютерной графики: учебное пособие / Е.В. Шикин. – М.: Диалог – МИФИ, 2014. – 215 с.

52. Щукина, Г.И. Педагогические проблемы формирования познавательных интересов учащихся. – М.: Педагогика, 2013. – 204 с.

53. Шауцукова, Л.З. Информатика: учебник для 10 – 11классов / – М.: Просвещение, 2010. – 256 с.

ПРИЛОЖЕНИЕ А MainForm

```
unit Main;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, Menus,
  StdCtrls, Dialogs, Buttons, Messages, ExtCtrls, ComCtrls, StdActns,
  ActnList, ToolWin, ImgList, AboutFormUnit;

type

  TMainForm = class(TForm)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    FileNewItem: TMenuItem;
    FileOpenItem: TMenuItem;
    FileCloseItem: TMenuItem;
    Window1: TMenuItem;
    Help1: TMenuItem;
    N1: TMenuItem;
    FileExitItem: TMenuItem;
    WindowCascadeItem: TMenuItem;
    WindowTileItem: TMenuItem;
    HelpAboutItem: TMenuItem;
    FileSaveItem: TMenuItem;
    FileSaveAsItem: TMenuItem;
    WindowMinimizeItem: TMenuItem;
    StatusBar: TStatusBar;
    WindowTileItem2: TMenuItem;
    ToolBar1: TToolBar;
    tbNew: TToolButton;
    tbOpen: TToolButton;
```

tbSave: TToolButton;
ToolButton15: TToolButton;
tbPointer: TToolButton;
tbAddVertex: TToolButton;
tbAddArc: TToolButton;
tbAddRebro: TToolButton;
tbDestroyArc: TToolButton;
tbDestroyRebro: TToolButton;
tbChangeArcWeight: TToolButton;
tbChangeRebroWeight: TToolButton;
tbFindTree: TToolButton;
tbFindWay: TToolButton;
ToolButton26: TToolButton;
ILmnu: TImageList;
ActionList2: TActionList;
acNew: TAction;
acOpen: TAction;
OpenDialog: TOpenDialog;
acSave: TAction;
acSaveAs: TAction;
SaveDialog: TSaveDialog;
acClose: TWindowClose;
acExit: TAction;
WindowArrange1: TWindowArrange;
acWCascade: TWindowCascade;
acWMinimize: TWindowMinimizeAll;
acWHor: TWindowTileHorizontal;
acWVert: TWindowTileVertical;
acAbout: TAction;
acPointer: TAction;
acAddVertex: TAction;
acAddArc: TAction;
acAddRebro: TAction;
acDestroyArc: TAction;

```
acDestroyRebro: TAction;
acChangeArcWeight: TAction;
acChangeRebroWeight: TAction;
acFindTree: TAction;
acFindWay: TAction;
N2: TMenuItem;
acSettings: TAction;
N3: TMenuItem;
acSelectAll: TAction;
N4: TMenuItem;
N5: TMenuItem;
acDelete: TAction;
N6: TMenuItem;
acStatistic: TAction;
N7: TMenuItem;
acShowSmeg: TAction;
N8: TMenuItem;
N9: TMenuItem;
acShowInced: TAction;
N10: TMenuItem;
acIzomorf: TAction;
N11: TMenuItem;
acSmegGraf: TAction;
N12: TMenuItem;
acHelp: TAction;
N13: TMenuItem;
procedure acNewExecute(Sender: TObject);
procedure acOpenExecute(Sender: TObject);
procedure acSaveExecute(Sender: TObject);
procedure acSaveAsExecute(Sender: TObject);
procedure acCloseExecute(Sender: TObject);
procedure acExitExecute(Sender: TObject);
procedure acAboutExecute(Sender: TObject);
procedure acPointerExecute(Sender: TObject);
```

```
    procedure acAddVertexExecute(Sender: TObject);
    procedure EnableActions;
    procedure DisableActions;
    procedure acAddArcExecute(Sender: TObject);
    procedure acAddRebroExecute(Sender: TObject);
    procedure acDestroyArcExecute(Sender: TObject);
    procedure acDestroyRebroExecute(Sender: TObject);
    procedure acChangeArcWeightExecute(Sender: TObject);
    procedure acChangeRebroWeightExecute(Sender: TObject);
    procedure acFindTreeExecute(Sender: TObject);
    procedure acFindWayExecute(Sender: TObject);
    procedure acSettingsExecute(Sender: TObject);
    procedure acSelectAllExecute(Sender: TObject);
    procedure acDeleteExecute(Sender: TObject);
    procedure acStatisticExecute(Sender: TObject);
    procedure acShowSmegExecute(Sender: TObject);
    procedure acShowIncedExecute(Sender: TObject);
    procedure acIzomorfExecute(Sender: TObject);
    procedure acSmegGrafExecute(Sender: TObject);
    procedure acHelpExecute(Sender: TObject);
  private
  public
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

uses ChildWin, IzomorfFormUnit;

procedure TMainForm.EnableActions;
```

```
begin
  if ActiveMDIChild<>nil then
  begin
    acPointer.Enabled:=true;
    acAddVertex.Enabled:=true;
    acSave.Enabled:=true;
    acSaveAs.Enabled:=true;
    acAddArc.Enabled:=true;
    acAddRebro.Enabled:=true;
    acDestroyArc.Enabled:=true;
    acDestroyRebro.Enabled:=true;
    acChangeArcWeight.Enabled:=true;
    acChangeRebroWeight.Enabled:=true;
    acFindTree.Enabled:=true;
    acFindWay.Enabled:=true;
    acSettings.Enabled:=true;
    acSettings.Enabled:=true;
    acSelectAll.Enabled:=true;
    acDelete.Enabled:=true;
    acStatistic.Enabled:=true;
    acShowSmeg.Enabled:=true;
    //acShowInced.Enabled:=true;
    acIzomorf.Enabled:=true;
    acSmegGraf.Enabled:=true;
  end;
end;

procedure TMainForm.DisableActions;
begin
  if (ActiveMDIChild=nil)or(ActiveMDIChild.MDIChildCount=0) then
begin
acPointer.Enabled:=false;
acAddVertex.Enabled:=false;
acSave.Enabled:=false;
```

```
acSaveAs.Enabled:=false; acAddArc.Enabled:=false;
acAddRebro.Enabled:=false;
acDestroyArc.Enabled:=false;
acDestroyRebro.Enabled:=false;
acChangeArcWeight.Enabled:=false;
acChangeRebroWeight.Enabled:=false;
acFindTree.Enabled:=false;
acFindWay.Enabled:=false;
acSettings.Enabled:=false;
acSettings.Enabled:=false;
acSelectAll.Enabled:=false;
acDelete.Enabled:=false;
acStatistic.Enabled:=false;
acShowSmeg.Enabled:=false;
//acShowInced.Enabled:=false;
acIzomorf.Enabled:=false;
acSmegGraf.Enabled:=false;
end;
end;

procedure TMainForm.acNewExecute(Sender: TObject);
var Child: TMDIChild;
begin
Child:=TMDIChild.Create(Self);
Child.Caption:='Без имени';
EnableActions;
end;

procedure TMainForm.acOpenExecute(Sender: TObject);
var Child: TMDIChild;
begin
if OpenDialog.Execute then
begin
Child:=TMDIChild.Create(Self);
```

```
Child.Caption:=OpenDialog.FileName;
if not(Child.Open(OpenDialog.FileName)) then
DisableActions
else
EnableActions;
end;
end;

procedure TMainForm.acSaveExecute(Sender: TObject);
begin
  MDIChild.Save('');
end;

procedure TMainForm.acSaveAsExecute(Sender: TObject);
begin
if SaveDialog.Execute then
MDIChild.Save(SaveDialog.FileName);
end;

procedure TMainForm.acCloseExecute(Sender: TObject);
begin
  if ActiveMDIChild<>nil then ActiveMDIChild.Close;
DisableActions;
end;

procedure TMainForm.acExitExecute(Sender: TObject);
begin
Close;
end;

procedure TMainForm.acAboutExecute(Sender: TObject);
begin
try
Application.CreateForm(TAboutForm, AboutForm);
```

```
AboutForm.ShowModal;
finally
AboutForm.Free;
end;
end;

procedure TMainForm.acPointerExecute(Sender: TObject);
begin
MDIChild.tbPointer;
end;

procedure TMainForm.acAddVertexExecute(Sender: TObject);
begin
MDIChild.tbAddVertex;
end;

procedure TMainForm.acAddArcExecute(Sender: TObject);
begin
 MDIChild.tbAddArc;
end;

procedure TMainForm.acAddRebroExecute(Sender: TObject);
begin
MDIChild.tbAddRebro;
end;

procedure TMainForm.acDestroyArcExecute(Sender: TObject);
begin
MDIChild.tbDestroyArc;
end;

procedure TMainForm.acDestroyRebroExecute(Sender: TObject);
begin
MDIChild.tbDestroyRebro;
```

```
end;

procedure TMainForm.acChangeArcWeightExecute(Sender: TObject);
begin
MDIChild.tbChangeArcWeight;
end;

procedure TMainForm.acChangeRebroWeightExecute(Sender: TObject);
begin
MDIChild.tbChangeRebroWeight;
end;

procedure TMainForm.acFindTreeExecute(Sender: TObject);
begin
MDIChild.tbFindTree;
end;

procedure TMainForm.acFindWayExecute(Sender: TObject);
begin
  MDIChild.tbFindWay;
end;

procedure TMainForm.acSettingsExecute(Sender: TObject);
begin
  MDIChild.tbSettings;
end;

procedure TMainForm.acSelectAllExecute(Sender: TObject);
begin
MDIChild.SelectAll;
end;

procedure TMainForm.acDeleteExecute(Sender: TObject);
begin
```

```
 if MDIChild.SelCount>0 then
 MDIChild.MIDeleteClick(Self);
end;

procedure TMainForm.acStatisticExecute(Sender: TObject);
begin
MDIChild.tbStatistic;
end;

procedure TMainForm.acShowSmegExecute(Sender: TObject);
begin
  acShowSmeg.Checked:=not(acShowSmeg.Checked);
  MDIChild.tbShowSmeg(acShowSmeg.Checked);
end;

procedure TMainForm.acShowIncedExecute(Sender: TObject);
begin
  acShowInced.Checked:=not(acShowInced.Checked);
MDIChild.tbShowInced(acShowInced.Checked);
end;

procedure TMainForm.acIzomorfExecute(Sender: TObject);
begin
Izomorf;
end;

procedure TMainForm.acSmegGrafExecute(Sender: TObject);
begin
 MDIChild.tbSmegGraf;
end;

procedure TMainForm.acHelpExecute(Sender: TObject);
begin
WinExec(PChar('hh.exe '+ExtractFilePath(Application.ExeName)+'Help.chm'), 1);
```

```
end;

end.
```

ПРИЛОЖЕНИЕ Б SmegGrafForm

```
unit SmegGrafFormUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Spin, Grids, Buttons, Childwin;

type
  TSmegGrafForm = class(TForm)
    SeNum: TSpinEdit;
    Label1: TLabel;
    SG: TStringGrid;
    BtnOk: TSpeedButton;
    BtnCancel: TSpeedButton;
    procedure SeNumChange(Sender: TObject);
    procedure SGKeyPress(Sender: TObject; var Key: Char);
    procedure BtnOkClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SGSetEditText(Sender: TObject; ACol, ARow: Integer;
      const Value: String);
    procedure BtnCancelClick(Sender: TObject);
  private
  public
  end;

var
  SmegGrafForm: TSmegGrafForm;

procedure SetSmeg;

implementation

uses Main;

{$R *.DFM}

procedure SetSmeg;
begin


Application.CreateForm(TSmegGrafForm, SmegGrafForm);
    SmegGrafForm.ShowModal;
  finally
```

```
    SmegGrafForm.Free;
  end;
end;

procedure TSmegGrafForm.SeNumChange(Sender: TObject);
var i:integer;
begin
  try
    SG.RowCount:=SeNum.Value+1;
    SG.ColCount:=SeNum.Value+1;
    for i:=1 to SG.RowCount-1 do
      SG.Rows[i].Strings[0]:=IntToStr(i);
    for i:=1 to SG.ColCount-1 do
      SG.Cols[i].Strings[0]:=IntToStr(i);
  except
  end;
end;

procedure TSmegGrafForm.SGKeyPress(Sender: TObject; var Key: Char);
begin
  if (Key>#32)and(((Key<>'0')and(Key<>'1'))or
     ((Length(SG.Rows[SG.Row].Strings[SG.Col])>0))) then Key:=#0;
end;

procedure TSmegGrafForm.BtnOkClick(Sender: TObject);
var MDI:TMDIChild;
    i,j:integer;
    angl:Double;
    w,h:Double;
    t,v1,v2:TVertexPtr;
    a:TArcPtr;
    UW:Boolean;
begin
  MDI:=TMDIChild(MainForm.ActiveMDIChild);
  MDI.RedR:=false;
  DestroyGraf(MDI.FirstVertex);
  MDI.SGS.RowCount:=1;
  MDI.SGS.ColCount:=1;
  MDI.FirstVertex:=nil;
  angl:=pi*2/SeNum.Value;
  w:=(MDI.ImgGraf.Width)/2-20;
  h:=(MDI.ImgGraf.Height)/2-20;
  for i:=1 to SeNum.Value do
  begin

t:=MDI.AddVertex(round(w+10+cos(angl*i)*w),round(h+10+sin(angl*i)*h),false,IntToStr(i));
    t^.Number:=i;
```

```
    t.LName.Caption:=IntToStr(i);
  end;
  MDI.LastNumber:=SeNum.Value;
  UW:=MDI.UserWeight;
  MDI.UserWeight:=false;
  for i:=1 to SeNum.Value do
  for j:=1 to i do
  begin
    if SG.Rows[i].Strings[j]='1' then
    begin
      v1:=MDI.GetVertex(i);
      v2:=MDI.GetVertex(j);
      a:=MDI.AddArc(v1,v2);
      a^.Arc:=false;
      a^.Weight:=1;
      a:=MDI.AddArc(v2,v1);
      a^.Arc:=false;
      a^.Weight:=1;
    end;
  end;
  MDI.UserWeight:=UW;
  MDI.RedR:=true;
  MDI.ReDrawGraf(true);
  Close;
end;

procedure TSmegGrafForm.FormCreate(Sender: TObject);
begin
  SG.Rows[1].Strings[0]:='1';
  SG.Cols[1].Strings[0]:='1';
end;

procedure TSmegGrafForm.SGSetEditText(Sender: TObject; ACol, ARow: Integer;
  const Value: String);
begin
  SG.Rows[ACol].Strings[ARow]:=Value;
end;

procedure TSmegGrafForm.BtnCancelClick(Sender: TObject);
begin
  Close;
end;

end.
```

ПРИЛОЖЕНИЕ В SetarcWeightForm

```
unit SetarcWeightFormUnit;

interface

uses
  Windows, Messages, SysUtils,Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TSetarcWeightForm = class(TForm)
    LMDSimpleLabel1: TLabel;
    WeightEdit: TEdit;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    Entered:Boolean;                    {Если нажата кнопка OK}
  end;

var
  SetarcWeightForm: TSetarcWeightForm;      {Эта форма}

implementation

{$R *.dfm}

  {Нажатие кнопки OK}
procedure TSetarcWeightForm.Button1Click(Sender: TObject);
```

```
begin
    try
      StrToFloat(WeightEdit.Text);
      Entered:=True;
      Close;
    except
      ShowMessage('Вес введён неверно!');
    end;
end;

procedure TSetarcWeightForm.FormShow(Sender: TObject);
begin
  Entered:=False;
  FocusControl(WeightEdit);
  WeightEdit.SelectAll;
end;

end.
```