ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ФАКУЛЬТЕТ МАТЕМАТИКИ И ЕСТЕСТВЕННОНАУЧНОГО ОБРАЗОВАНИЯ

КАФЕДРА ИНФОРМАТИКИ, ЕСТЕСТВЕННОНАУЧНЫХ ДИСЦИПЛИН И МЕТОДИК ПРЕПОДАВАНИЯ

ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ ТЕОРИИ ГРАФОВ НА УРОКАХ ИНФОРМАТИКИ В ШКОЛЕ

Выпускная квалификационная работа обучающегося
по направлению подготовки
44.04.01 Педагогическое образование
магистерская программа Информационные технологии в образовании
заочной формы обучения, группы 02041561
Кривенко Веры Александровны

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

Рецензент
директор МБОУ «СОШ №24»
г. Белгорода им. Героя Советского
Союза И.П. Крамчанинова
Конюхова В.И.

БЕЛГОРОД 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный этап развития общества характеризуется сильным влиянием на него компьютерных и информационных технологий, которые проникают во все сферы человеческой деятельности.

Теория графов, начало которой было положено Эйлером в его знаменитом рассуждении о Кёнигсбергских мостах (1736 г.), сегодня понятия и утверждения теории графов широко применяется в математике, физике, электронике, экономике, программировании и других научных и прикладных областях. В виде графов можно представлять дороги между населенными пунктами, электрические схемы, строение химических молекул, отношения между людьми и решать разнообразные задачи и головоломки; задания на использование графов содержатся в материалах международных исследований образовательных достижений учащихся PISA [1].

В целом, анализируя учебные программы университетов, можно с уверенностью говорить, что учащиеся физико-математических, механико-математических и других факультетов связанных с изучением математики, активно изучают теорию графов как отдельный курс. Данные курсы помогают освоить многочисленную теорию, связанную с графами, но самое главное они показывают все разнообразие применения графов при решении конкретных задач из реальной жизни. Не смотря на то, что в школьной программе изучение теории графов явно не предусмотрено, некоторые положения теории графов включены в обязательный минимум содержания основных образовательных программ по предмету «Информатике и ИКТ» [3].

Актуальность темы выпускной квалификационной работы определяется тем, что отдельный пласт предлагаемых в школьных учебниках и сборниках задач, заданий из ОГЭ и ЕГЭ можно красиво и доступно решать с учащимися, используя графы, некоторые из них «требуют» применения графов [30]. Это позволяет расширить спектр средств, используемых при решении задач, упростить поиск решения и в некоторых случаях существенно сократить время

на поиск правильного ответа. Именно это и вызывает интерес у учащихся к практическим задачам. Данный интерес и стал основой для дальнейшего продвижения графов на уроках информатики.

Объектом исследования выпускной квалификационной работы является применение теории графов в информатике.

Предметом исследования выпускной квалификационной работы является программа для обучения решению задач по информатике с использованием теории графов.

Цель выпускной квалификационной работы состоит в особенности использования теории графов на уроках информатики в школе.

Для решения поставленной цели необходимо решить ряд задач:

- изучить и проанализировать учебную, методическую литературу по данной теме;
- сравнить подходы к решению задач без использования и с использованием графов;
- подобрать задачный материал по данной теме;
- разработать приложение для решения задач с использованием теории графов.

Практическая значимость – предложенные исследования и программа может быть использована в учебном процессе при изучении темы «Теория графов»; при подготовке к ОГЭ и ЕГЭ.

Выпускная квалификационная работа состоит из введения, заключения, трех разделов, списка использованных источников и приложения.

Введение содержит общие сведения о работе, актуальность выбранной темы, объект, цель и задачи. В первой главе демонстрируется аналитический обзор этапов развития теории графов. Вторая глава включает в себя анализ учебно-методической литературы, описание алгоритмов и их применение.

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

1 АНАЛИТИЧЕСКИЙ ОБЗОР

В последнее время графы и связанные с ними методы исследований органически пронизывают на разных уровнях едва ли не всю современную математику. Графы эффективно используются в теории планирования и управления, теории расписаний, социологии, экономике, биологии, медицине, географии [10]. Широкое применение находят графы в таких областях, как программирование, электроника, в решении вероятностных и комбинаторных задач, нахождения кратчайшего расстояния, максимального паросочетания и др. Математические развлечения и головоломки тоже являются частью теории графов. Теория графов быстро развивается, находит все новые приложения.

Любая система, предполагающая наличие дискретных состояний или наличие узлов и переходов между ними может быть описана графом. Язык графов оказывается удобным для описания многих физических, технических, экономических, биологических, социальных и других систем.

Теория графов находит применение, например, в геоинформационных системах (ГИС). Существующие или вновь проектируемые дома, сооружения, кварталы и т.п. рассматриваются как вершины, а соединяющие их дороги, инженерные сети, линии электропередач и т.п. – как рёбра. Применение различных вычислений, производимых на таком графе, позволяет, например, найти кратчайший объездной путь или ближайший продуктовый магазин, спланировать оптимальный маршрут [9].

1.1 Этапы развития графов

Истоками тории графов стало не одно изобретение, а целых три, и создавались они в любом из направлений своеобразно. Само же составление их как теории относится к 1936 году, когда венгерским математиком Денешем Кенигом (1884 –1944) была размещена книга «Теория конечных и бесконечных

графов» [8]. К подлинному времени они имеют большое количество приложений, в всевозможных научных дисциплинах, таких как: химия и физика, а еще во множества иных.

Одним из первых итогов в теории графов явился аспект существования всех ребер графа без повторений, приобретенный Леонардом Эйлером (1707−1783) – швейцарским, германским и русским математиком и механиком, внесшим базовый вклад в становление данных наук. В 1736 г. научного работника привлекла задачка о 7 мостах германского мегаполиса Кенигсберга (ныне Калининград), о чем он написал в собственном послании к итальянскому арифметику Д.Д. Маринони от 13 марта 1736 года. В данном послании Эйлер сообщает о том, собственно, что ему была предложена задачка о полуострове, расположенном в мегаполисе Кенигсберге, и окруженном речкой Прегель, сквозь которую перекинуто 7 мостов. Спрашивается, имеет возможность ли кто-либо беспрерывно обогнуть их, проходя лишь только однажды сквозь любой мост. В соответствии с рисунком 1.1 задачку о Кенигсбергских мостах схематически возможно представить так.

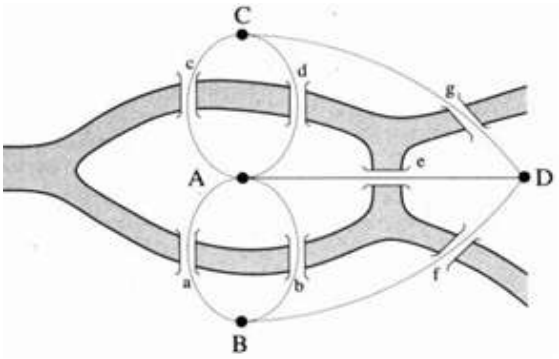

Рисунок 1.1 – Схема Кёнигсбергских мостов

Думая над задачей, Эйлер пришел к выводам:

- число нечетных вершин всякий раз четно;
- одним росчерком пера возможно начертить граф, вершины которого четные, при этом начать возможно из всякой вершины графа и

окончить в ней;

- граф с 2-мя нечетными вершинами, возможно, начертить одним росчерком пера, при этом начинать в одной нечетной вершине, а окончить в совершенно другой;
- граф больше чем с 2-мя нечетными вершинами нельзя начертить одним росчерком.

У графа кенигсбергских мостов 4 нечетных вершины. Вследствие этого, пройти по всем мостам, ни по одному не проходя два раза, не видится вероятным. Граф, который возможно обогнуть, проходя раз один по любому из его ребер, называется уникурсальным. Это же определение содержит и уникурсальная фигура.

Но Эйлер и принял решение для ряда задач, основанных на теории графов, все же немаловажные шаги в данной области были изготовлены только в XIX веке. Всемирно знакомый германский инженер-электрик Густав Роберт Кирхгоф (1824−1887) содействовал данному. Ученый отыскал в неявной форме количество остовов случайного, данного связного графа, а означает, что количество помеченных деревьев. Этим образом, при составлении абсолютной системы уравнений для токов и напряжений в электрической схеме, Кирхгоф внес предложение по существу представлять эту схему графом и отыскивать в этом графе остовные деревья с поддержкой которых отличаются линейно независимые системы контуров электрических цепей. Эти графы принято именовать сигнальными. В соответствии с рисунком 1.2 представлена электрическая цепь, соответствующий ей граф и остовное дерево.

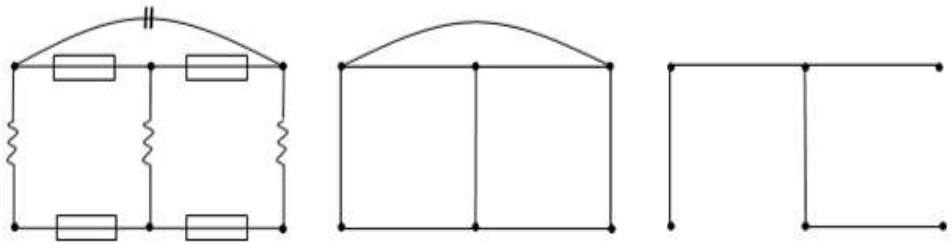

Рисунок 1.2 – Электрическая схема Кирхгофа

Важно для теории графов использования к задаче анализ структуры химических соединений. Имеющая место быть в химической практики графическое изображение структура молекул сформировалась в 60-е годы XIX века. Особое внимание вызвала неувязка изомерии, то есть комментарий ситуации, когда молекулы препаратов имеют один и тот же состав, но разные свойства. Данная проблема была решена графически.

В 1875 году Артур Кэли опубликовал работу в «Berichte der deutschen Chemischen Gesellschaft» по перечислению алкановых изомеров. Данная работа практически считалась первой по использованию теории графов в химии. Естественно, ученый определил задачу абстрактно: отыскать количество всех деревьев с p вершинами, любое из которых содержит вершину со степенями равными единице и четырем.

К концу XIX века способы заключения задачи на графах в работах Кэли пополнились надлежащими достижениями:

- решение задачи о перечисление деревьев, степень вершин которых не превосходят четыре;
- решение задачи о перечисление деревьев, степень вершин которых равны единицы и четырем;
- решение задачи о перечисление всех деревьев;
- решение задачи о перечисление корневых деревьев.

Существует еще один вид задач, связанных с путешествиями вдоль графов. Речь идет 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 XIX века 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 XIX века. 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].

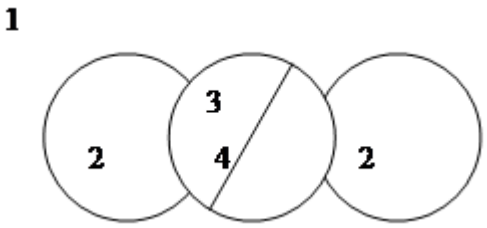

Рисунок 1.3 – Схематическое изображение задачи о четырех красках

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 необходимо провести газ, свет и воду.

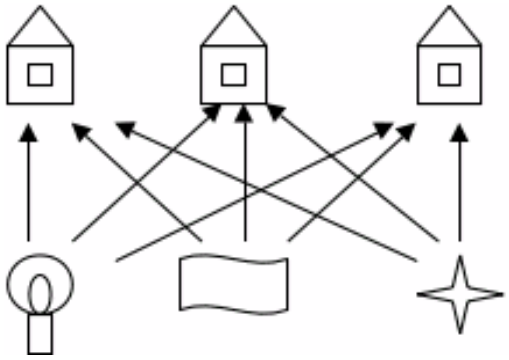

Рисунок 1.4 – Задача об электро, газо и водоснабжении

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.

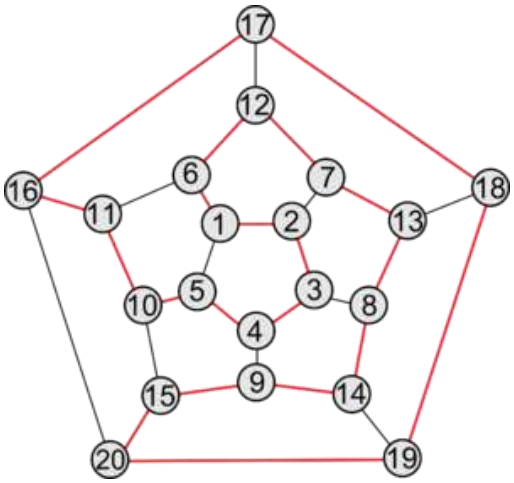

Рисунок 1.5 – Теорема Куратовского

В XX веке 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 XX века 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 20-30-х годах XX века 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].

1.2 Основные понятия теории графов

Графом G(V,E) называется совокупность двух множеств – непустого множества V (вершин) и множества E двухэлементных подмножеств множества V (E – множество ребер). Графически граф может быть представлен геометрической фигурой, в которой вершина изображена точкой, а ребро – отрезком линии, соединяющей точки, соответствующие концевым вершинам ребра. Элементы множеств V и E могут содержать индексы. Индексы вершин обозначают их номера, индексы ребер – номера соединяемых ими вершин. Запись e_{ij} означает, что ребро графа образовано парой вершин v_i и v_j. Например, если множества $V = \{v_1,v_2,v_3,v_4,v_5\}$ и $E = \{e_{12},e_{13},e_{14},e_{15},e_{24},e_{25},e_{35}\}$, то граф G(V,E) может быть представлен в соответствии с рисунком 1.6.

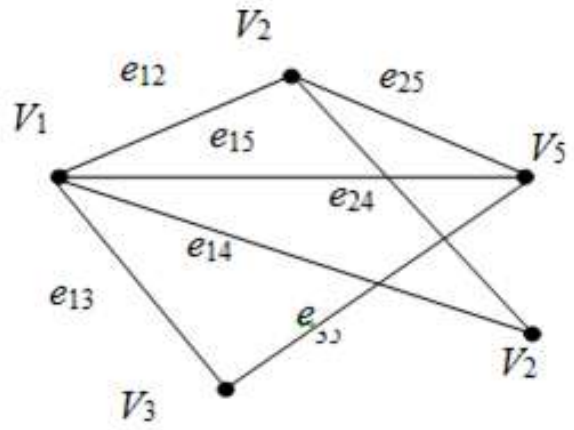

Рисунок 1.6 – Граф

Любой из графов может содержать одно или несколько ребер, у которых оба конца сходятся в одной вершине. Такие ребра называются петлями. Граф с петлями называют псевдографом. В соответствии с рисунком 1.7 представлен граф с петлями (псевдограф).

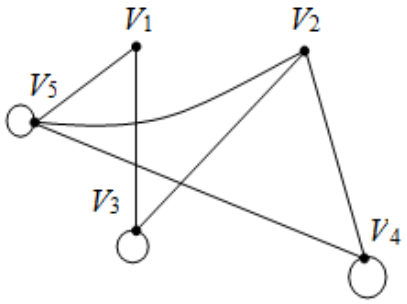

Рисунок 1.7 – Псевдограф

Если ребрам графа приданы направления от одной вершины к другой, то такой граф называется ориентированным. Ребра ориентированного графа называются дугами. Соответствующие вершины ориентированного графа называют началом и концом. Пример ориентированного графа представлен на рисунке 1.8. Если направления ребер не указываются, то граф называется неориентированным или просто. Ребра такого графа иногда называются звеньями. Для неориентированного графа используется понятие инцидентного ребра. Пусть V1 и V2 – вершины, e12 – соединяющее их ребро. Тогда ребро e12 инцидентно данным вершинам и наоборот, вершины V1 и V2 инцидентны ребру e12. Два ребра инцидентные одной вершине, называются смежными.

В ряде случаев рассматриваются смешанные графы, имеющие как ориентированные, так и неориентированные ребра.

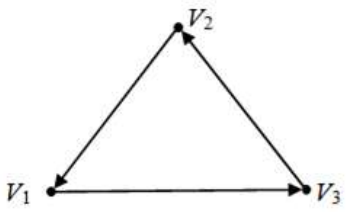

Рисунок 1.8 – Ор-граф

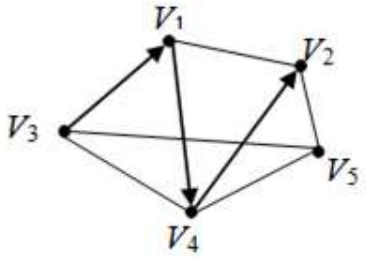

Рисунок 1.9 – Смешанный граф

Пара вершин может соединяться с двумя и более ребрами дугами, такие ребра (дуги) называются кратными. Граф, не имеющий ни ребер, ни вершин называется пустым. Такой граф обозначается GØ. Граф, состоящий только из изолированных вершин, то есть не содержащий ни одного ребра называется, нуль-графом. Такой граф обозначается G0.

Простым графом называют граф, который не имеет петель или кратных ребер. Полный граф – это простой граф, у которого любые две вершины соединены ребром. Полный граф обозначается GП. Полный ориентированный граф называется турниром.

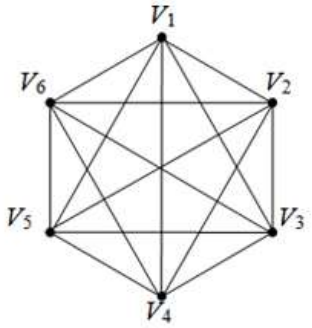

Рисунок 1.10 – Полный граф

Плотный граф – это полный граф, у которого при каждой вершине имеется петля. Плотный граф обозначается G'П. Граф, состоящий из одной вершины, называется тривиальным.

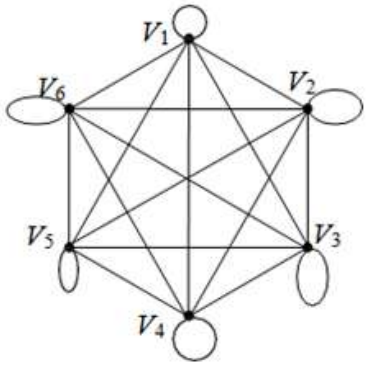

Рисунок 1.11 – Плотный граф

Следующий граф, который мы рассмотрим − плоский, который нарисован на плоскости так, что все его ребра пересекаются только в его вершинах. Любой граф, изоморфный плоскому, называют планарным графом. Плоский граф есть изображение планарного, но не каждое изображение планарного графа является плоским.

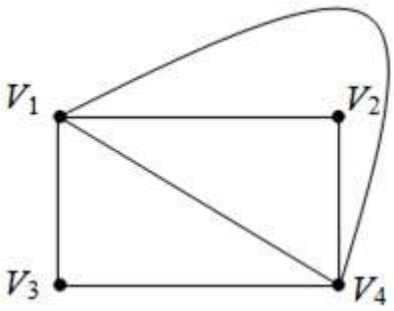

Рисунок 1.12 – Плоский граф

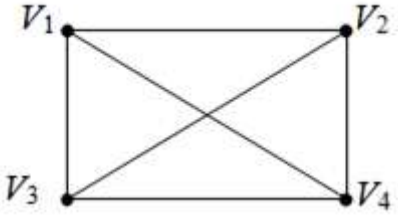

Рисунок 1.13 – Планарный граф

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].

Разберем понятие связности графа. Итак, граф считается связным, если из любой его вершины есть путь (маршрут) в любую другую вершину (путь может состоять из любого количества ребер). Пример связности приведен на рисунке 1.14. Однако, если, например, удалить ребро между вершинами 4 и 5, то граф не будет связным, так как из вершины 5 нельзя будет попасть ни в какую другую вершину.

Путь есть маршрут в орграфе без повторяющихся дуг, простой путь – без повторяющихся вершин. Если существует путь из одной вершины в другую, то вторая вершина достижима из первой. На рисунке 1.14 приведен пример пути.

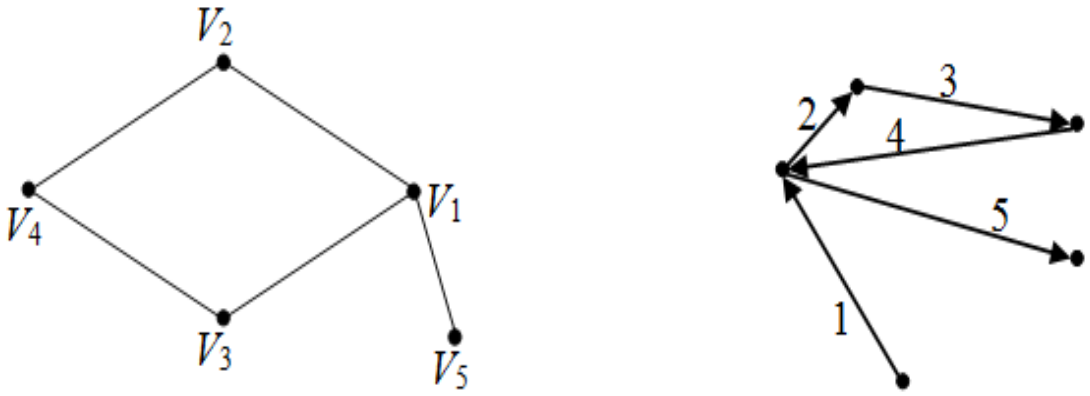

Рисунок 1.14 – Связный граф и путь в ор-графе

До сих пор граф G предполагался неориентированным. Для графа можно вводить как неориентированные маршруты, цепи и простые цепи, так и ориентированные. Теперь обратимся к ор-графу. Связность ор-графов определяется в принципе так же, как и n-графов. Специфичным для ор-графа (или смешанного графа) является понятие сильной связности.

Ор-граф называется сильно-связным, если любые его вершины взаимно достижимы. Ор-граф называется слабо-связным, если является связным n-граф, полученный из него заменой ориентированных ребер неориентированными. Граф, полученный заменой дуг на ребра, называется ассоциированным ор-графу.

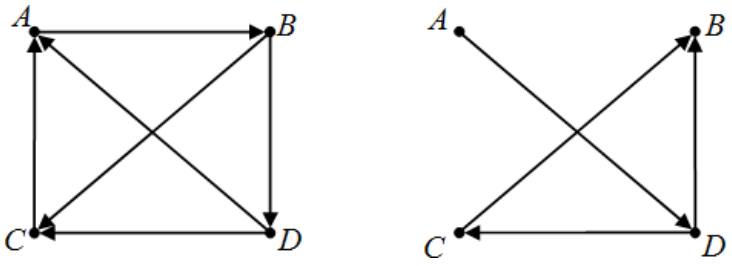

Рисунок 1.15 – Сильно-связный граф и слабосвязанный граф

Любой связный неориентированный граф является сильно связным. Поэтому существует такое разбиение множества вершин графа на попарно

непересекающиеся подмножества, что все вершины в каждом подмножестве связаны, а вершины из различных подмножеств не связаны. Каждое такое подмножество вершин графа вместе с ребрами, инцидентными этим вершинам, образует связный подграф. Следовательно, неориентированный граф представим единственным образом в виде непересекающихся связных подграфов. Эти подграфы называют компонентами связности.

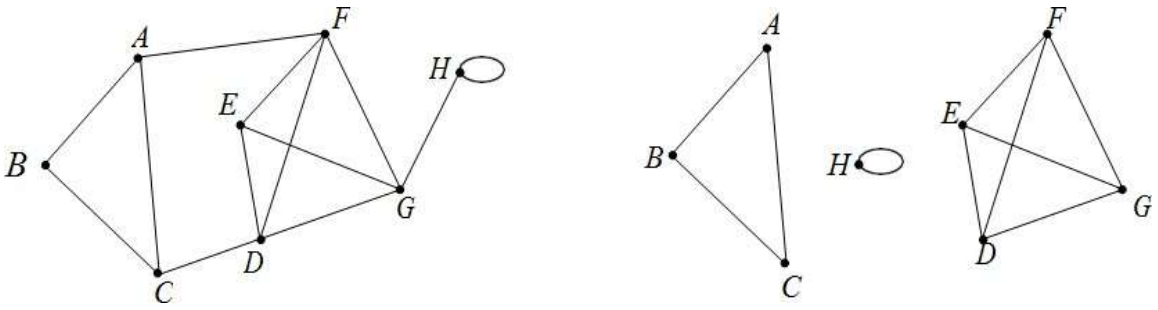

Рисунок 1.16 – Граф и разбиение графа на компоненты

Дерево – это конечный связный граф с выделенной вершиной (корнем) без циклов. Дерево не имеет петель и кратных ребер. Связность означает наличие путей между любой парой вершин, а отсутствие циклов значит, то, что между парами вершин имеется только по одному пути. Наличие этих двух свойств позволяет жестко связать число вершин и число ребер: в дереве с n вершинами всегда n–1 ребер.

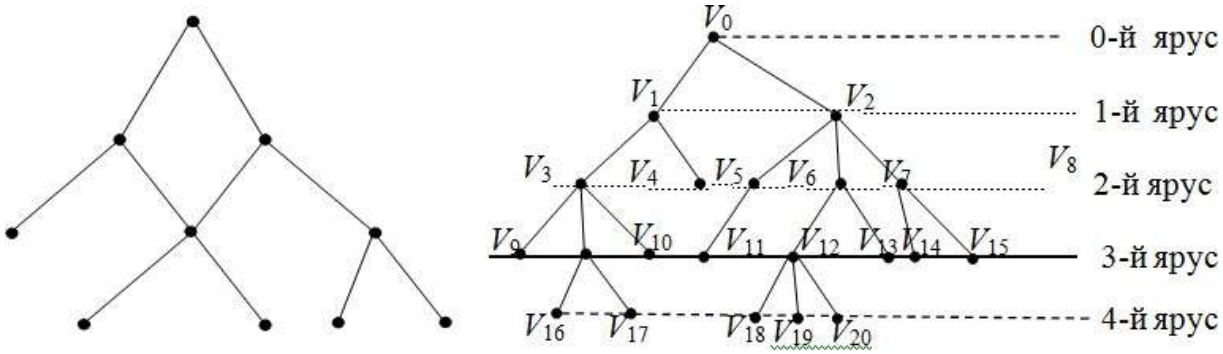

Рисунок 1.17 – Граф -дерево

Вершина графа-дерева называется висячей, если ее степень равна 1. Ребро, инцидентное концевой вершине, называется концевым. Если конечное дерево состоит более чем из одной вершины, оно имеет хотя бы две концевые вершины и хотя бы одно концевое ребро. Ориентированным деревом называется дерево имеющее корень, а все остальные вершины – листья. Вершина с нулевой степенью захода называется корнем дерева, вершины с нулевой степенью исхода (из которых не исходит ни одна дуга) называются листьями.

При описании ориентированных деревьев принято использовать термины: отец, сын, предок, потомок. Каждая вершина дерева называется узлом.

Упорядоченным деревом называется дерево, в котором поддеревья каждого узла образуют упорядоченное подмножество. Для упорядоченных деревьев принята терминология: старший и младший сын для обозначения соответственно первого и последнего сыновей некоторого узла.

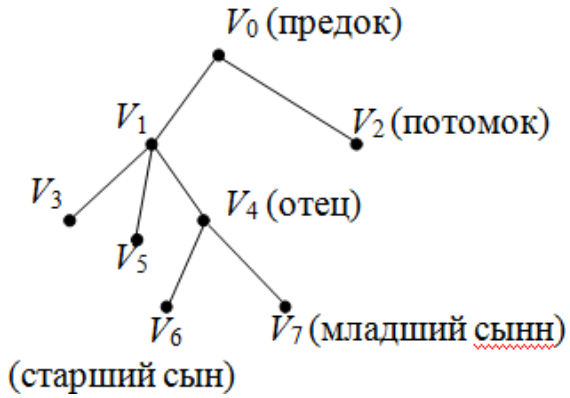

Рисунок 1.18 – Упорядоченное дерево

Лесом называется множество, содержащее несколько непересекающихся графов-деревьев.

Рисунок 1.19 – Пример множества лес

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.

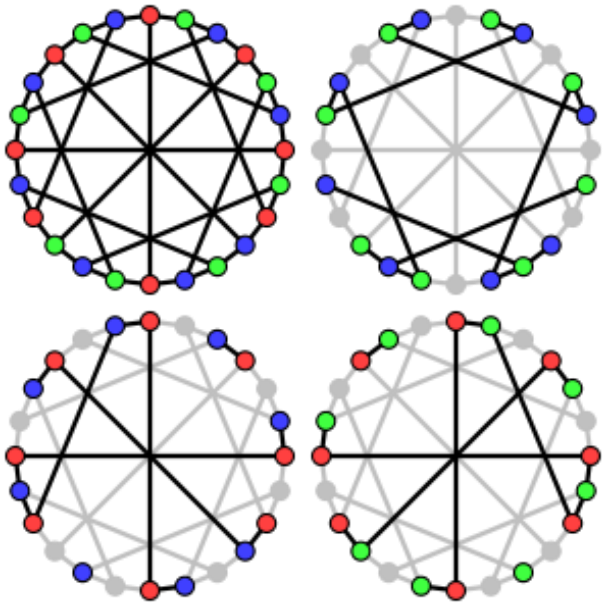

Рисунок 1.20 – Раскраска графов

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:

$$D[u] + A[u, v] < D[v], \text{ оценку } D[v] \text{ улучшаем: } D[v] = D[u] + A[u, v] \qquad (1)$$

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 [25].

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 [21]. Опишем теперь более детально методы нахождения расстояния от фиксированной вершины, называемой источником, его всегда будем обозначать через S, до всех остальных вершин графа.

Для общего случая представим алгоритм, в котором подразумевается только лишь отсутствие контуров с отрицательной протяженностью. С данным алгоритмом как правило связывают имена Р.Е. Беллмана и Л.Р. Форда.

Следует отметить, что при изучении информатики и математики ученики постоянно встречаются с простыми понятиями и задачами теории графов: задача коммивояжера, нахождение кратчайших путей в сетях, проблема четырех красок, представление алгоритма в виде блок-схемы, отображение иерархии данных в виде дерева. Связывание логических, текстовых, комбинаторных задач с использованием графов моделей.

2 ПОДБОР ЗАДАЧНОГО МАТЕРИАЛЛА, ОПИСАНИЕ МОДЕЛИ ПРИЛОЖЕНИЯ

2.1 Анализ учебно-методической литературы курса «Информатика и ИКТ»

Проанализируем учебники и учебные пособия по информатике для более точной картины представления учебного материала по школьному курсу «Информатика и ИКТ»:

1. Учебник Н. Д. Угринович, Информатика и ИКТ 9 класс.

Рисунок 2.1 – Учебник «Информатика и ИКТ» Н. Д. Угринович

Содержание и объём учебного материала данного учебника соответствует Государственному стандарту и обязательному минимуму содержания образования по информатике; в учебнике автор ставит вопросы для самоконтроля.

В данном учебнике есть задачи для закрепления теоретического материала, но их недостаточно; так же вскользь представлено несколько заданий по теории графов, тема, на наш взгляд, не раскрывается. Задачи

составлены в соответствии с теоретическим материалом и расположены с нарастанием трудности их решения.

В учебнике нет задач с занимательным содержанием. Задания с историческим содержанием используются при изучение темы моделирование и формализация. Учебный материал изложен доступным языком, достаточно просто и убедительно. Например, как записывается число в позиционной системе счисления.

2. Учебник Босова Л.Л., Босова А.Ю. Информатика и ИКТ для 9 класса в 2-х частях.

Рисунок 2.2 – Учебник «Информатика и ИКТ» Босова Л.Л., Босова А.Ю.

В учебнике кроме основной информации содержатся многочисленные ссылки на образовательные ресурсы. На страницах данного учебника подробно рассмотрены решения типовых задач по каждой тем. В конце глав приведены тестовые задания, которые помогут ученику оценить, хорошо ли он освоил теоретический материал и сможет ли применять свои знания для решения задач на практике. Что касается темы о графах, то как и в предыдущем учебнике, на

наш взгляд, тема раскрывается очень слабо, представлены теоретические вопросы.

Учебник иллюстрирован необходимыми графиками, рисунками, чертежами, схемами. Иллюстрации расположены согласно изложению теоретического материала. Качество иллюстраций – удовлетворительное. Учебник включен в список рекомендованных учебников министерством образования, в качестве практикум, методическое пособие 7 – 11 классы.

3. Быкадоров Ю.А. Информатика и ИКТ. 9 класс.

Рисунок 2.3 – Учебник «Информатика и ИКТ» Быкадоров Ю.А

Учебник включен в список рекомендованных учебников МО. Содержание учебника «Информатика и ИКТ» для 9 класса соответствует Федеральному государственному образовательному стандарту основного общего образования по информатике и ИКТ.

Чтобы поддержать и расширить интерес учащихся к информатике, автор учебника построил изложение материала на основе системы заданий практической направленности, отражающих проблемные ситуации, возникающие в процессе использования компьютера.

Структура материала учебников нацелена на реализацию принципа индивидуализации обучения. С учетом этого материал учебников реализует индивидуальные методы обучения на уроках информатики в форме, аналогичной лабораторным работам, когда учащиеся пользуются учебником как справочным руководством, основная часть урока отводится на самостоятельную работу школьников, а учитель выступает в роли «постановщика задач» и консультанта.

Упражнения в учебниках снабжены пошаговым описанием хода их выполнения, что облегчает работу слабым учащимся, однако предусмотрен и широкий набор заданий повышенной сложности, предназначенных для сильных учащихся, для которых учебники выполняют функции задачника и справочника по типовым операциям обработки информации, и именно в данном учебнике подробно представлена тема по теории графов.

Кроме того, в учебнике для 9 класса добавлен обширный материал по программированию на языке Паскаль в дополнение к языку JavaScript.

В электронных приложениях, размещенных на сайте издательства в свободном доступе, помещены рабочие материалы для выполнения разобранных в учебнике упражнений и практических заданий для самостоятельного выполнения. Кроме них, для каждого занятия предоставляются демонстрационные материалы в форме презентаций Power Point, где приведены и наглядно проиллюстрированы основные определения по каждой теме, показаны базовые приемы работы в изучаемых программах, даны цветные образцы для выполнения упражнений и практических заданий, связанных с графическими построениями.

Электронная форма учебников дает возможность организовать как самостоятельную, так и групповую работу учащихся, в результате чего у них накапливается опыт сотрудничества в процессе учебной деятельности.

Методические пособия содержат тематическое планирование, комментарии к главам учебника, дополнительные задания, тесты и

контрольные работы, что существенно сокращает время подготовки учителя к уроку.

Межпредметные связи курса информатики в данном учебнике реализуются в темах: кодирование, обработка числовой информации, моделирование и формализация. В учебнике большое внимание уделяется формированию у учащихся алгоритмического и системного мышления, а также практических умений и навыков в области информационных технологий.

Но, на наш взгляд, в нем недостаточно материала для подготовки к ЕГЭ. А очень хорошо даются практические работы с применением всевозможных программных средств. Задания даются с учётом уровня развития школьников. Многие из программ, которые разбираются в практической части, есть в приложении к методичке по учебнику.

Проанализируем далее методические пособия по информатике:

4. М. С. Цветкова, О. Б. Богомолова ИНФОРМАТИКА. УМК для основной школы 7– 9 классы. Методическое пособие для учителя.

Рисунок 2.4 – «Информатика. УМК для основной школы»

Методическое пособие содержит методические рекомендации в соответствии с требованиями ФГОС для планирования, организации обучения в новой информационной среде школы. Представлены содержание учебного предмета, описание УМК, тематическое и поурочное планирование по курсу информатики для 7–9 классов, таблицы соответствия УМК требованиям ФГОС, планируемые результаты обучения, а так же раздел «Электронное приложение к УМК», описывающий электронную форму учебников «Электронный УМК» (www.eumk.Lbz.ru). Стоит сказать, что здесь освещена тема систем, моделей и графов.

5. Босова А.Ю. «Информатика и ИКТ» для основной школы (5-9 классы) ограниченной ответственностью «БИНОМ. Лаборатория знаний».

Рисунок 2.5 – «Информатика и ИКТ» для основной школы

УМК отвечает всем современным требованиям и обеспечивает: развитие мотивационных, операциональных и когнитивных личностных ресурсов учащихся; формирование ИКТ-компетентности и подготовку школьников к сдаче ОГЭ; подготовку молодых людей к жизни и продолжению образования в современном высокотехнологичном мире.

Особенностью УМК является то, что он приемлем для обучения, даже если в начальной школе предмет не велся. Завершенность: для основного

общего образования – да, для среднего общего образования – нет. В состав также входят электронные приложения: электронные формы учебников, электронное методическое приложение и мультимедийные объекты в их составе:

- методические материалы для учителя;
- файлы-заготовки (тексты, изображения), необходимые для выполнения работ компьютерного практикума;
- текстовые файлы с дидактическими материалами (для печати);
- дополнительные материалы для чтения;
- мультимедийные презентации ко всем параграфам каждого из учебников;
- интерактивные тесты и электронное методическое приложение.

6. Семакин И.Г., Залогова Л.А., Русаков С.В., Шестакова Л.В «Информатика и ИКТ» 7-9 классы. Общество с ограниченной ответственностью «БИНОМ. Лаборатория знаний».

Рисунок 2.6 – «Информатика и ИКТ» Семакин И Г.

Основная цель – решение задачи формирования школьного курса информатики как полноценного общеобразовательного предмета. В содержании этого предмета отражены три составляющие предметной (и образовательной) области информатики:

- теоретическая информатика;
- прикладная информатика (средства информатизации и информационные технологии);
- социальная информатика.

Технологическая составляющая курса усилена. Это связано с изменением названия предмета, произошедшего в 2004 г. (с «Информатика» на «Информатика и ИКТ»), так и с концепцией образовательного стандарта.

В содержании курса выдержан принцип инвариантности к конкретным моделям компьютеров и версиям программного обеспечения. Упор делается на понимание идей и принципов, заложенных в информационных технологиях, а не на последовательности манипуляций в средах конкретных программных продуктов.

Важно, что в учебниках параллельно рассматриваются операционная система Windows и ее приложения, а также свободно распространяемая операционная система Linux и ее приложения.

На сегодняшний день Единый государственный экзамен (ЕГЭ) является единственной формой государственной итоговой аттестации выпускников школ. Одним из необязательных предметов (предметов по выбору) для сдачи Единого государственного экзамена является информатика и ИКТ.

Рисунок 2.7 – Пример сборника по подготовке к ЕГЭ

Общее количество заданий (27) и максимальный первичный балл 35. Каждый тестовый вариант состоит из двух частей:

- часть 1 содержит 23 задания (1-23) с кратким ответом в виде числа, последовательности букв или цифр, записанных без пробелов и других разделителей;
- часть 2 содержит 4 задания (24–27) с развернутым ответом, требующим записи полного решения задания на отдельном бланке ответов.

Каждое из заданий части 1 оценивается одним первичным баллом, тогда как задания части 2 оцениваются от двух до четырех первичных баллов. По уровню сложности задания распределены: базовый – задания 1-12; повышенный – задания 13-22, 24; высокий – задания 23, 25-27.

На основании мнений экспертов предметной комиссии по проверке заданий с развернутым ответом, можно выделить ряд проблем в подготовке учащихся. Слабая подготовка участников экзамена в области программирования и алгоритмизации, проявляющаяся в неспособности «видеть алгоритм целиком», определить результат выполнения алгоритма, найти существенную ошибку в алгоритме и исправить её (задания 24-25).

Возникновение трудностей при составлении алгоритма: учащиеся путаются в условиях, в границах массива, неверно организуют цикл. При описании алгоритма на естественном языке остаются проблемы с точностью формулировок. Учащиеся не справляются с заданиям, требующим проведение анализа алгоритма (задания 21, 24).

В целом можно предположить, что причиной низких результатов выполнения заданий является:

- «бескомпьютерный» вариант выполнения;
- отсутствие этой темы в программе базового курса информатики старшей школе;
- «неравномерность» изучения тем школьного курса информатики в различных образовательных учреждениях.

Таким образом, анализ ЕГЭ по информатике и ИКТ позволит спроектировать предупреждающие и корректирующие мероприятия для совершенствования практики обучения в области информатики и ИКТ и повышения его качества.

Экзамен по информатике и ИКТ в 9 классе лежит в компетенции выбора ученика. Школьники выбирают данное испытание по желанию. Охват тем на экзамене по информатике и ИКТ в 9 классе повторяет тематику ЕГЭ по предмету в 11 классе. В связи с этим залог успешности при итоговой аттестации после обучения в старших классах закладывается во многом уже в средней школе.

Рисунок 2.8 – Пример сборника по подготовке к ОГЭ (ГИА)

Тематика охватывает знания учащихся по кодированию информации, системам счисления, основам логики, математическому моделированию, алгоритмизации и программированию, информационным технологиям работы с электронными таблицами, базами данных и в сети Интернет. Экзаменационная работа состоит из двух частей:

- часть 1 содержит 18 заданий базового и повышенного уровней сложности, среди которых 6 заданий с выбором и записью ответа в виде одной цифры и 12 заданий, подразумевающих самостоятельное формулирование и запись экзаменуемым ответа в виде последовательности символов;
- часть 2 содержит 2 задания высокого уровня сложности. Задания этой части подразумевают практическую работу учащихся за компьютером с использованием специального программного обеспечения. Результатом исполнения каждого задания является отдельный файл. Задание 20 дается в двух вариантах: 20.1 и 20.2; экзаменуемый должен выбрать один из вариантов задания.

Распределение заданий КИМ по уровням сложности:

- Часть 1 экзаменационной работы содержит 11 заданий базового уровня сложности и 7 заданий повышенного уровня сложности;

- Часть 2 содержит 2 задания высокого уровня сложности.

Анализ данных показывает, что наибольшую сложность у выпускников вызывают задания:

- на выполнение умения исполнить алгоритм, записанный на естественном языке;
- обрабатывать цепочки символов или списки;
- умения анализировать информацию, представленную в виде схем;
- умения исполнить алгоритм для конкретного исполнителя с фиксированным набором команд;

Таким образом, можно сделать вывод о том, что учащиеся средней школы осознанно выбирают экзамен по информатике и ИКТ. Это подтверждает в целом эффективность организации процесса обучения информатике, что позволяет получать ученикам фундаментальную подготовку в данной предметной области.

Вместе с тем вопросы изучения элементов теории графов так и не нашли широкого внедрения в учебный процесс школы, в частности, школьную математику. Это было обусловлено тем, что не хватало времени на уроках математики, не существовало эффективных средств для изучения теории графов во внеурочное время. Поэтому изучение этих вопросов долгое время оставалось лишь фрагментарным.

2.2 Алгоритмы поиска пути в графах

Поиск в глубину (DFS).

Алгоритм поиска в глубину описывается следующим образом: для каждой непройденной вершины необходимо найти все непройденные смежные вершины и повторить поиск для них. Используется в качестве подпрограммы в алгоритмах поиска одно- и двусвязных компонент, топологической сортировки.

Реализуется проще BFS, но затрат на ресурсов больше, так как здесь главную роль играет рекурсия

Алгоритм Дейкстры

Находит кратчайшее расстояние от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса (так как на таком цикле бесконечно будет уменьшатся наилучший путь). На каждом шаге цикла мы ищем вершину с минимальным расстоянием и флагом равным нулю. Затем мы отмечаем её пройденной и проверяем все соседние с ней вершины. Если в ней расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его. Цикл завершается когда все вершины будут пройдены. Время работы алгоритма оценивается как O(N^2).

Алгоритм Флойда.

Также находит массив кратчайших расстояний. Но в отличие от алгоритма Дейкстры, он использует динамическое программирование. На каждом шаге цикла k создаётся массив решений, где w[i,j] содержит минимальное расстояние, где используется используются только вершины 1,2..k и сами i и j. На начальном этапе W копирует матрицу смежности. Тогда на каждом k есть два варианта — минимальный путь идёт через вершину k или нет. Теоретически такой метод гораздо легче реализовать (банальный перебор), но использует больше машинных ресурсов, чем Дейкстра (сложность алгоритма оценивается как O(N^3), но зато ищет минимальные пути между всеми парами точек).

Алгоритм Краскала.

Находит каркас минимального веса, т.е такой подграф исходного графа, который бы был связным, содержал все вершины исходного графа и суммарный вес рёбер был наименьшим. В этой задаче используется список рёбер. Вначале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла (т.е.

зачем добавлять ребро R(i,j) в подграф, который содержит эти вершины, а значит, от одной можно добраться до другой), выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Массив рёбер должен быть заранее отсортирован во весу (можно привести док-во: если сначала рассматривать ребро R1(i,j)>R2(i,j), то он потом будет удалён, так как мы встретили в списке рёбер R2(i,j), который весит меньше R1, а удаление рёбер в алгоритме не предусматривается).

Поиск в ширину (однопоточная версия).

Суть алгоритма заключается в том, чтобы перебрать всех преемников начальной вершины (корневого узла), и дальше по цепочке. Такой алгоритм помогает получить компоненту связности, то есть схему, куда можно прийти из какой-то заданной вершины. Применяя этот алгоритм поочерёдно для всех вершин, можно найти кратчайшее расстояние, оптимальный путь между двумя вершинами и так далее, в зависимости от предложенных условий.

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.

2.3 Описание интерфейса

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.

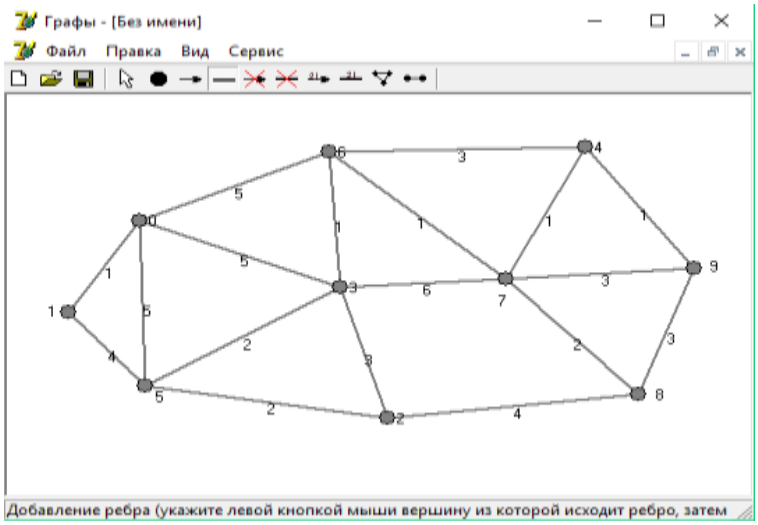

Рисунок 2.9 – Дерево графа

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.

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).

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.

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).

Кратчайший путь будет найден на основе тех алгоритмов, которые были изложены выше. Результат которые будет выведен продемонстрирован на рисунке 2.10.

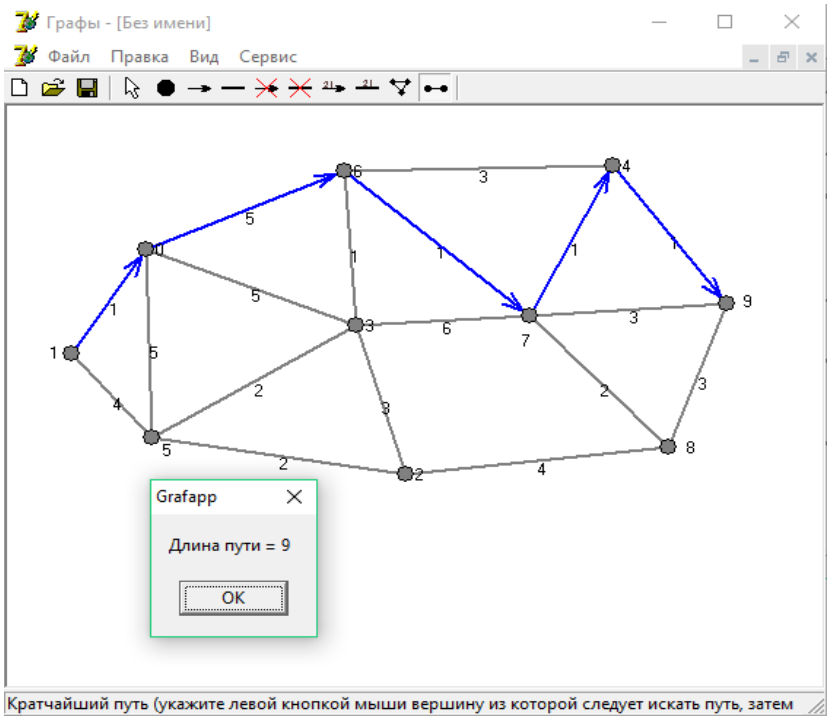

Рисунок 2.10 – Нахождение минимального пути

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 ПРИЛОЖЕНИЯ

3.1 Использование теории графов на уроках информатики

Сейчас теория графов стала очень популярной среди учителей, школьников и студентов. Это связанно с тем, что при помощи этой теории можно довольно просто решать большой круг самых разнообразных математических задач. На языке теории графов условия задач приобретают наглядность, что упрощает их анализ.

Сами решения, как правило, являются простыми, и, в отличие от решений другими методами, не содержат утомительных вычислений. Это является очевидным достоинством, так как изобилие выкладок е свидетельствует о содержательности теории. Теория графов притягательна как раз тем, что при своей наглядности и простате помогает решать самые серьезные математические и прикладные задачи.

Однако, как это ни странно, теория графов не входит в учебные планы математических специальностей университетов и в программы школ. В лучшем случае она изучается студентами и школьниками лишь на факультативах и спецкурсах.

Проблема изучения учениками поиска решения трудных математических задач, считается одной из наиболее сложных в теории и методике обучения математике и информатике.

Проявляется она, прежде всего в том, что математически способные учащиеся не имеют возможности отыскать способы решения, зная и понимая весь необходимый для этого теоретический материал.

Важным для каждого преподавателя считается вопрос о том, какие ресурсы применять при обучении. Средства преподавания (обучения) стали не только источником учебной информации, так и инструментом управления познавательной деятельностью подростков.

Средства обучения должны содействовать освоению основ дисциплин, формированию мышления, развитию мировоззрения, воспитанию учащихся в духе нравственности. Необходимо чтобы они были приспособлены к труду педагога и обучающихся, обеспечивать использование эффективных методов и приемов работы [16].

Решая проблемы при исследовании компонентов теории графов, следует не забывать, что в любом шаге, на любом этапе ее решения следует использовать творческий процесс. С самого начала, на первоначальном этапе, оно состоит в том, найти способ проанализировать и зашифровать условия задачи.

Второй этап - схематическая запись, складывается из геометрического представлении графов, и на данной стадии компонент творчества весьма значим вследствие того, что далеко не просто найти соответствия среди элементами условия и соответствующими компонентами графа.

Все другие этапы тоже не обходятся без применения творчества и находчивости. Осуществление поиска метода и реализации решения задачи (с проверкой и исследованием) имеет необходимость в последующих способностях решающих: умение абстрагирования, умение прогнозирования, способность гибкого применения теории графов, умение использования всех распространенных математических способов решения. Безусловно, построение ответа задачи – это также творческое изобретение, т.к. кроме того нужна и кодировка, и абстрагирование.

Облегчение восприятия и освоение обучающимися математических познаний могут быть достигнуты целесообразным применением различных средств наглядности – таблиц, чертежей и рисунков и т. д.

Рассмотрим некоторые средства наглядности с целью обучения поиску решения задач по теме «Теория графов».

Задача 1. В поселке 9 домов. Известно, что у Ивана соседи Петр и Антон, Максим сосед Петру и Сергею, Дима – Виктору и Никите, Евгений – сосед Никиты, а больше соседей в поселке нет (соседними считаются дома, у которых

есть общий участок забора). Может ли Иван огородами пробраться к Никите за грушами?

Решением этой 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 Ивана к дому Никиты нельзя [29].

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.

Рассмотрение теоретических положений и доказательства теорем осуществляются учащимися с опорой на наглядность и интуитивную составляющую мышления. Для старшеклассников можно рассмотреть некоторые темы для занятий. К основным понятиям теории графов целесообразно подводить школьников, решая задачи.

Примеры тем для изучения:

- Абстрактное определение графа;
- Определение геометрического графа;
- Инцидентность и смежность элементов графа;
- Степени вершин графа;
- Изоморфизм графов;
- Геометрическая реализация абстрактных графов;
- Части графа;
- Непрерывные последовательности ребер графа;
- Связность графов;
- Множества сочленения и разделяющие множества;
- Уникурсальные графы;
- Гамильтоновы графы.

3.2 Описание программной реализации

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

Таблица 1 – Описание переменных функций

Наименование	Тип	Назначение
MainForm	string	Ввод, вывод информации
SmegGrafForm	string	Представление графа в виде матрицы смежности
SetarcWeightFormUnit	string	Вес расстояния между вершинами графа

Форма Main Form располагает в себе главный интерфейс программы, где располагаются элементы управления и осуществляется ввод и вывод информации. Так же содержит в себе процедуры, описанные в других формах. Данная функция реализована следующим фрагментом кода:

```
unit Main;
interface
type
…
procedure acSaveAs_Execute (Sender: TObject);
procedure ac_NewExecute (Sender: TObject);
procedure Disable_Actions;
procedure ac_OpenExecute (Sender: TObject);
procedure acClose_Execute (Sender: TObject);
procedure acAbout_Execute (Sender: TObject);
procedure acPointer_Execute (Sender: TObject);
procedure acExit_Execute (Sender: TObject);
procedure acAdd_Vertex_Execute (Sender: TObject);
```

```
procedure acSave_Execute (Sender: TObject);
procedure Enable_Actions;
procedure acDestroy_ArcExecute (Sender: TObject);
procedure acAdd_Arc_Execute (Sender: TObject);
procedure acAddRebro_Execute (Sender: TObject);
procedure acDestroy_Rebro_Execute (Sender: TObject);
procedure acChangeArc_WeightExecute (Sender: TObject);
procedure acShowInced_Execute (Sender: TObject);
procedure acIzomorf_Execute (Sender: TObject);
procedure acSmegGraf_Execute (Sender: TObject);
procedure acFind_WayExecute (Sender: TObject);
procedure acSettings_Execute (Sender: TObject);
procedure ac_SelectAllExecute (Sender: TObject);
procedure ac_DeleteExecute (Sender: TObject);
procedure acChangeRebro_Weight_Execute (Sender: TObject);
procedure acFindTree_Execute (Sender: TObject);
procedure acStatistic_Execute (Sender: TObject);
procedure acShow_SmegExecute (Sender: TObject);
procedure acHelpExecute (Sender: TObject);
private
public
end;
```

Форма Smeg Graf Form несет ответственность за визуализацию графа в виде матрицы смежности. Для создания нового графа необходимо задать его указав вершины и расстояние между ними, получив матрицу смежности.. Пример реализации команды выглядит следующим образом:

```
unit Smeg_Graf_FormUnit;
intirface
uses
…
```

```
type
…
procedure Btn_OkClick (Sender: TObject);
procedure SeNum_Change (Sender: TObject);
procedure Form_Create (Sender: TObject);
procedure SG_KeyPress (Sender: TObject; var Key: Char);

procedure SGSet_EditText (ACol, ARow: Integer; Sender: TObject; const
Value: String);
procedure Btn_Cancel_Click (Sender: TObject);
private
public
end;
```

Форма Setarc Weight Form Unit хранит вес расстояния между вершинами графа. В нее попадают данные после построения вершин пользователем, соединив вершины графов ненаправленными или направленными маршрутами. Данная функция реализована следующим фрагментом кода:

```
unit SitarcWeieghtFormUnit;
interfece
uses
…
Type
…
procedure ForShow(Sender: TObject);
procedure Button1_Click(Sender: TObject);
private
{ Public declarations }{
public
Private declarations }
Entared:Boolean;                    {Если нажата кнопка OK}
```

end;

Данные в программу вводятся 3 способами. Первый способ ввода данных осуществляется при помощи файла.

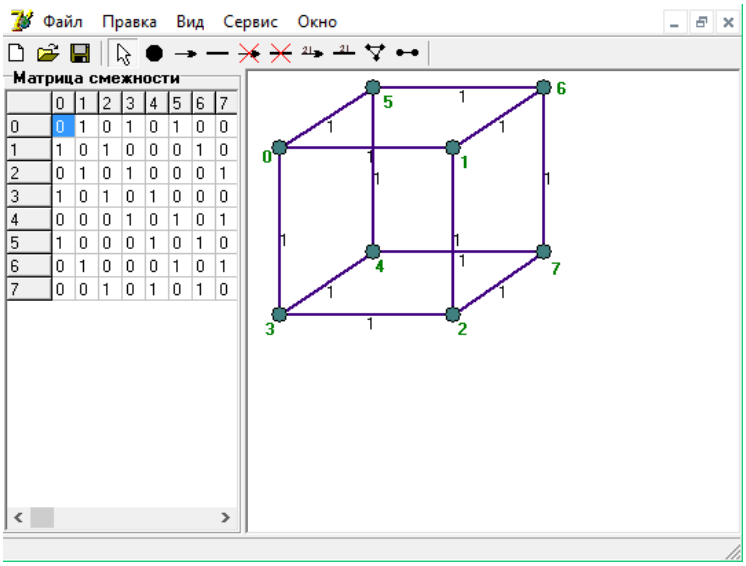

Рисунок 3.1 – Ввод данных из файла

Второй способ задания графа при помощи заполнения матрицы смежности внутри самой программы.

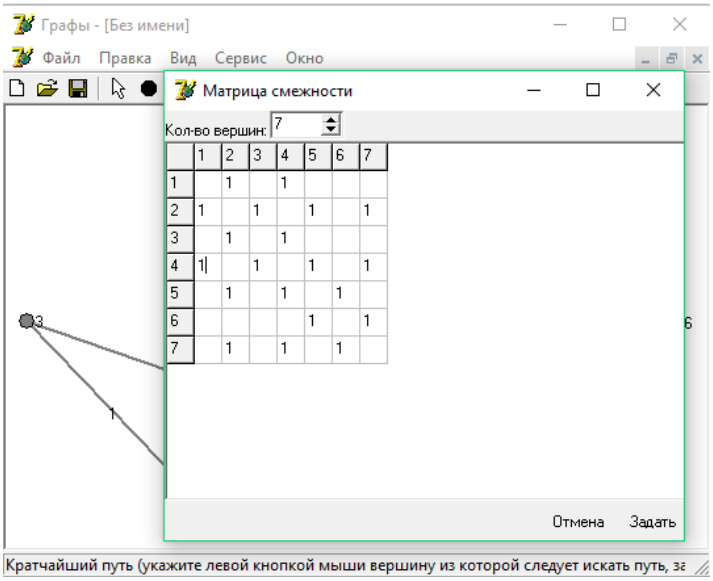

Рисунок 3.2 – Ввод данных через матрицу смежности

Третий способ задания графа это произвольная расстановка вершин и их соединение с указанием веса пути.

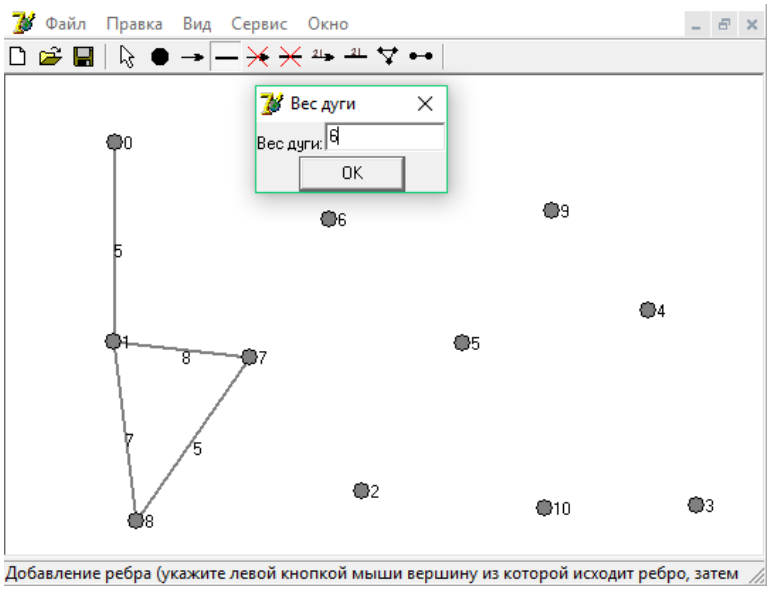

Рисунок 3.3 – Ввод данных через произвольную расстановку вершин

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

Моделирование с помощью графов реализует одну из важнейших потребностей – потребность наглядности. Визуализация модели явления в сочетании с вычислительными, информационными и моделирующими возможностями компьютера лучше всего объясняет сущность изучаемого явления. Рисунок графа является знаком, материальным предметом, который чувственно воспринимается и выступает в качестве посредника между реальной действительностью и математической моделью. Использование рисунков графов неразрывно связано с процессами абстрагирования и детализации, с помощью которых происходит отделение тех признаков моделируемого объекта, и которые затем отображаются в модели. Графу модели обеспечивают связь мышления с реальными ситуациями.

В рамках выполнения выпускной квалификационной работы была собрана информация об использовании теории графов в школьном курсе «Информатика и ИКТ».

Были рассмотрены теоретические основы, необходимые для изучения теории графов. Нами были рассмотрены следующие аспекты: возникновение теории графов как отдельной науки, основные виды графов, связность, графы-деревья, способы задания и операции на графах.

Проведен анализ учебной и методической литературы по теме исследования. Выяснены цель, задачи содержание обучения учащихся основной школы по теме: «Теория графов» было выявлено, что наиболее полно раскрывается данная тема в учебно-методическом комплексе под редакцией Быкадоров Ю.А., «Информатика и ИКТ».

Разработан программный продукт, который может быть внедрен и использован для изучения теории графов на уроках информатики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Агапов Р. С. О трех поколениях компьютерных технологий обучения в школе // Информатика и образование, 2014. 52 с.

2. Алексеев В. В. Физическое и математическое моделирование – СПб.: Питер, 1992. – 368 с.

3. Басова Л. Л. Информатика и ИКТ : учебник для 9 класса / Л. Л. Басова. – Москва : БИНОМ. Лаборатория знаний, 2012. – 244 с.

4. Басова Л. Л. Информатика. Методическое пособие / Л. Л. Басова, А. Ю. Басова. – Москва : БИНОМ. Лаборатория знаний, 2015. – 184 с.

5. Басова Л. Л., Басова А. Ю., Коломенская Ю. Г.. Занимательные задачи по информатике / – Москва: БИНОМ. Лаборатория знаний, 2014. – 164 с.

6. Басова Л.Л. Информатика. 7–9 классы / Л.Л. Басова, А.Б Басова – Москва: БИНОМ. Лаборатория знаний, 2013 – 84 с.

7. Белоусов А.В., Ткачев С.В.. Дискретная математика. – М.: Изд-во МГТУ им. Баумана, 2002.

8. Березина А.Ю. Графы и их применение / А.Ю. Березина. – Москва: Просвещение, 1979. – 143с.

9. Гейн А. Г. Информатика. Методические рекомендации / А. Т. Гейн. – Москва : Просвещение, 2013. – 80 с.

10. Гейн А. Т. Информатика. 11 класс / А. Т. Гейн. – Москва : Просвещение, 2014. – 336 с.

11. Глазунов С. А. Опорные конспекты как средство повышения качества образования / С. А. Глазунов – Москва: БИНОМ, 2013 – 211 с.

12. Ефимова О. Н. Курс компьютерной технологии с основами информатики. Учебное пособие для старших классов / О. Н. Ефимова, В. С. Морозов, Н. Г. Угринович. – Москва : ABF, 1999. – 432 с.

13. Калмыкова Н. В. Опорный конспект как один из способов представления учебной информации / Н. В. Калмыкова, С. Ф. Петряева – Молодой ученый: 2015. – №11.1. – 58 с.

14. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К.. Алгоритмы. Построение и анализ (3-е издание). – М.: Издательский дом «Вильямс», 2013.

15. Кузнецов А.А. Примерные программы по учебным предметам. Информатика. 7-9 классы. / Москва: Просвещение, 2012. – 102с.

16. Лапчик М. П. Методика преподавания информатики / М. П. Лапчик. – Москва : Академия, 2016. – 624 с.

17. Лыскова В. Ю. Учебные задачи в курсе информатики / В. Ю. Лыскова, У. Ф. Ракитина // Информатика и образование. – 1998. – №4. 61 с.

18. Макарова Н. В. Информатика и ИКТ : учебник для 8 – 9 классов / Н. В. Макарова. – СПб. : Питер, 2014. – 416 с.

19. Макарова Н. В. Информатика и ИКТ : учебник для 9 класса / Н. В. Макарова. – СПб. : Питер, 2001. – 254 с.

20. Мельников О. И. Незнайка в стране графов: Пособие для учащихся / О.И. Мельников – Москва: Наука, 2010. – 81 с.

21. Николаев А. С. Информатика 9 класс. Поурочные планы по учебнику Н. Д. Угриновича / А. С. Николаев. – Волгоград: Учитель, 2013. – 199 с.

22. Новиков А.А.. Дискретная математика для программистов. – СПб.: Питер, 2001

23. Оре О. Графы и их применение / О. Оре, – М: Мир, 1963 – 174 с.

24. Перминова Л.М. Образовательные стандарты в контексте школьного обучения / Л.М. Перминова – Москва: Просвещение, 2013.

25. Селиванов В. Л. Организация учебно–исследовательской работы студентов и школьников по информатике – Новосибирск: Перо, 2011. – 101 с.

26. Семакин И. Г. Информатика и информационные технологии. 10- 11 класс / И. Г. Семакин, Н. Д. Угринович – Москва: Бином, 2011. – 283 с. 32.

27. Семакин И. Г. Информатика. 9 класс / И. Г. Семакин, Е. К. Хеннер. – Москва: Лаборатория базовых знаний, 2015. – 224 с.

28. Семакин И. Г. Информатика. Программа для основной школы: 7–9 классы / И. Г. Семакин, М. С. Цветкова – Москва: БИНОМ. Лаборатория знаний, 2012. – 201 с.

29. Стандарт основного общего образования по информатике и информационным технологиям // Информатика и образование. – 2014.

30. Угринович Н. Д. Информатика и ИКТ. 10-11 класс / Н. Д. Угринович – Москва: БИНОМ. Лаборатория знаний, 2005. – 232 с.

31. Угринович Н. Д. Информатика. Программа для основной школы. ФГОС / Н. Д. Угринович, М. С. Цветкова, Н. Н. Самылкина. – Москва: БИНОМ. Лаборатория знаний, 2016. – 256 с.

32. Угринович Н. Д. Информатика: учебник для 11 класса / Н.Д. Угринович. – Москва: Бином. Лаборатория знаний, 2013. – 182 с.

33. Угринович Н.Д.. Информатика и ИКТ. Профильный уровень. Москва, БИНОМ. Лаборатория базовых знаний, 2008.

34. Уилсон Р.. Введение в теорию графов. – М.: Мир, 1977.

35. Федеральный государственный стандарт общего образования / Министерство образования и науки Российской Федерации. - Москва: Просвещение, 2014.

36. Федотова С. Г. Курс лекций по информатике. Учебное пособие / С. Г. Федотова. – Москва: Форум, 2016. – 485 с.

37. Филиппов В. И. Метапредметные результаты по информатике, достижение которых проверяется в ходе ГИА в форме ОГЭ и ЕГЭ // Конференциум АСОУ: сборник научных трудов и материалов научно-практических конференций. – Москва, 2015. – 738 с.

38. Фридланд А. Я. Информатика: процессы, системы, ресурсы / А. Я. Фридланд. – Москва: БИНОМ. Лаборатория знаний, 2012. – 272 с.

39. Фридман И. Научные методы в архитектуре / И. Фридман; пер. с англ. А.А. Воронова. – М.: Стройиздат, 1983. – 160 с.

40. Харари Ф. Теория Графов / Ф. Харари – Москва: Мир, 1979 – 298с.

41. Шершакова Т. А. Решение задач на движение и работу с помощью графов / Т.А. Шершаков – Москва, 1987. – 167 с.

42. Э. Майника. Алгоритмы оптимизации на сетях и графах. – М.: Мир, 1981.

ПРИЛОЖЕНИЕ А

Листинг приложения для решения задач с использованием теории графов

```
const MaxN = 50;
INF = 1000000000; //"бесконечность"
type Matrix = array[1..MaxN,1..MaxN] of longint; //тип матрицы смежности. M[i,j] = true, если существует ребро, идущее от вершины i к j
var
A : Matrix; N, S1, S2: integer;
input: text;
procedure Input_Table(var A : Matrix; N : longint; var T : Text); //процедура ввода матрицы смежности A(N, N) из текстового файла T
var i, j : longint;
begin
for i := 1 to N do
begin
for j := 1 to N do
begin
read(T, A[i, j]);
if (a[i,j] = 0) and (i <> j) then a[i,j] := INF; //вершины, которые не связаны ребром, будем обзначать "бесконечностью" ввиду ограничения на вес рёбер
end;
readln(T);
end;
end;
procedure Deikstr(s, s1 : integer); //s, s1 - искомые вершины (необходимо найти путь от s до s1)
var i, j, v, min, z : longint;
st, c : string;
visited : array[1..MaxN]of boolean; //массив посещённости вершин
```

D : array[1..MaxN] of longint; //массив кратчайших расстояний

P : array[1..MaxN] of integer; //массив предков, который поможет определить маршрут. p[i] будет содержать предпоследнюю вершину кратчайшего маршрута от s до i

```
begin
for i := 1 to N do
begin
p[i] := s;
visited[i] := FALSE;
end;
visited[s] := TRUE; //вершина S посещена
for i := 1 to N do D[i] := A[s, i]; //изначальный массив расстояний
D[s] := 0;
p[s] := 0; //
for i := 1 to N-1 do //на каждом шаге находим минимальное решение и пытаемся его улучшить
begin
min := INF;
for j := 1 to N do if (not visited[j]) and (D[j] < min) then
begin
min := D[j]; //минимальное расстояние
v := j; //найденная вершина
end;
for j := 1 to N do if (D[j] > D[v] + A[v, j]) and (D[v] < INF) and (A[v, j] < INF) then
begin
```

D[j] := D[v] + A[v, j]; //пытаемся улучшить решение. Если в ней расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его.

```
p[j] := v;
```

```
end;
s := v; //новая текущая вершина
visited[v] := TRUE; //и она отмечается посещенной
end;
st := ''; //осталось преобразовать в формат вывода (мы проидёмся по всем вершинам кратчайшего пути от s до s1, но только в обратном порядке)
z := p[s1]; //пока есть корневая вершина
while z <> 0 do
begin
str(z,c);
st := c + '->' + st; //заносим в маршрут
z := p[z]; //переходим к следующей вершине
end;
str(s1,c); //в маршрут записываем начальную вершину
st := st + c;
writeln(st);
end;
BEGIN
assign(input,'input.txt');
reset(input);
readln(input, N, S1, S2);
Input_Table(A, N, input);
close(input);
Deikstr(S1, S2);
END.
unit Main;
interface
uses Windows, Graphics, Forms, SysUtils, Classes, Controls, Menus, StdCtrls, Dialogs, ComCtrls, StdActns Buttons, Messages, ExtCtrls, ActnList, ToolWin, ImgList, AboutFormUnit;
```

```
type
FileOpenItem: TMenuItem;
MainMenu1: TMainMenu;
File1: TMenuItem;
TMainForm = class(TForm)
FileNewItem: TMenuItem;
FileCloseItem: TMenuItem;
Window1: TMenuItem;
Help1: TMenuItem;
N1: TMenuItem;
FileExitItem: TMenuItem;
StatusBar: TStatusBar;
WindowTileItem: TMenuItem;
HelpAboutItem: TMenuItem;
WindowCascadeItem: TMenuItem;
WindowMinimizeItem: TMenuItem;
tbNew: TToolButton;
WindowTileItem2: TMenuItem;
ToolBar1: TToolBar;
tbOpen: TToolButton;
tbSave: TToolButton;
ToolButton15: TToolButton;
tbAddArc: TToolButton;
tbAddRebro: TToolButton;
tbDestroyRebro: TToolButton;
tbChangeArcWeight: TToolButton;
tbChangeRebroWeight: TToolButton;
tbDestroyArc: 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:='Безимени';
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(Applicaon.ExeName)+'Help.chm'1);
end.

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),f
alse,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.

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;                                {ЕслинажатакнопкаОК}
end;
var
SetarcWeightForm: TSetarcWeightForm;                        {Этаформа}
implementation {$R *.dfm}                       {Нажатиекнопки ОК}
procedure TSetarcWeightForm.Button1Click(Sender: TObject);
begin
try
StrToFloat(WeightEdit.Text);
Entered:=True;
Close;
except
ShowMessage('Весвведённеверно!');
end;
procedure TSetarcWeightForm.FormShow(Sender: TObject);
begin
Entered:=False;
FocusControl(WeightEdit);
WeightEdit.SelectAll;
end.
```