Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Пермский национальный исследовательский политехнический университет
Лысьвенский филиал

Факультет профессионального образования
Направление 09.03.01 Информатика и вычислительная техника
Кафедра Естественнонаучных дисциплин

Зав. кафедрой ЕН
канд.физ.-мат.наук
_____/И.Т. Мухаметьянов
«____»_____2017 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на соискание академической степени бакалавра

На тему «Разработка приложения автоматизированного поиска денотатов в научно-технической литературе»

Студент: _____/ Е.В. Ерискина
(подпись) (Фамилия И.О.)

Состав ВКР:
1. Пояснительная записка на ___ стр.
2. Электронный носитель с материалами ВКР.

Руководитель ВКР: Канд.техн.наук, доц.
_____ Д.С. Курушин
(подпись, дата)

Лысьва, 2017 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Выделение главных мыслей из текста интересовало человека со времен появления письменности. В современном мире, с появлением Интернета эта тема приобрела новую актуальность из-за постоянно растущих объемов данных. Решение этой задачи полезно для различных сфер деятельности человека, например, для быстрого получения знаний о предметной области из огромного количества источников.

Одним из способов решения данной задачи является выделение денотатов.

Гипотеза работы строится на предположении о том, что в качестве модели понимания основной мысли текста, может быть использована такая модель его содержания, которую можно представить в виде денотатного графа. Такой граф отобразит иерархию денотатов и их отношений, что является графической моделью линейного текста.

Создание приложения автоматизированного поиска денотатов не является новой задачей, но до сих пор она остается нерешенной. Кроме практической направленности актуальность создания системы поиска денотатов обусловлена связью с такими проблемами, как текст и закономерности его восприятия и понимания, семантика языковых единиц текста и выделение основного содержания.

Объектом исследования выпускной квалификационной работы выступает денотативный анализ текста как аналитико-синтаксический процесс.

Цель выпускной квалификационной работы: исследовать структуру и методы денотативного анализа текстов и создать приложение для автоматического поиска денотатов.

Задачи выпускной квалификационной работы:

1. Изучить принципы реферативного перевода текста.
2. Построить модель данных, на основе обработанной информации.

3. На основе модели данных построить эталонный граф для выбранной предметной области, и с его помощью провести экспериментальный перевод текстов.

4. С помощью инфологической модели написать алгоритм для автоматизированного приложения поиска денотатов.

5. Протестировать полученный код посредством экспериментального извлечения денотатов из текста для выбранного скопоса.

6. Проанализировать полученные результаты, исправить недочеты и провести повторное тестирование.

1 ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

1.1 Реферативный перевода текста

Реферативный перевод можно определить одновременно и как аналитический и как синтаксический процесс, включающий в себя одновременно и перевод и реферирование. Уточним понятие «одновременности» перевода и реферирования. Перевод это такой процесс, который имеет недвусмысленный выраженный результат, зафиксированный средствами выходного языка в устной или письменной форме. То же самое можно сказать и реферировании, с той разницей, что этот процесс осуществляется в пределах одного и того же языка. Если принять, что T_1 и P_1 – текст и реферат на входном языке, а T_2 и P_2 - текст и реферат на выходном языке, то процесс типа $T_1 \rightarrow T_2$ – это перевод без реферирования, а $T_1 \rightarrow P_1$ – это реферирование без перевода. В случае же $T_1 \rightarrow P_1 \rightarrow P_2$ или $T_1 \rightarrow T_2 \rightarrow P_1$ имеют место и перевод и реферирование, осуществляемые в одном цикле. Однако, эти процессы не являются реферативным переводом. Реферативным переводом можно считать процесс $T_1 \rightarrow P_2$, особенностью которого является отсутствие промежуточного звена между исходными данными и конечным результатом. Можно предположить, что процесс реферирования осуществляется слитно, без промежуточного звена, но такому представлению противоречит следующий факт. Известно, что условие адекватного перевода и реферирования является глубокое понимание обрабатываемого текста. Именно понимание является общим между переводом и реферированием. Для каждого из этих процессов требуется различный уровень понимания. Если для перевода достаточно осмысления отдельных участков теста, то для реферирования необходимо понимания текста в целом. Только так можно оценить, что является в тексте главным, а что дополнительным. Отсюда следует, что установка не реферирование стимулирует такое осмысление текста, которые приводит к формированию в интеллекте переводчика мыслительного преобразования, представляющего тест в целом. Это означает, что в реферативном переводе все-таки существует промежуточное

звено. Это звено представляет собой результат осмысления и понимания текста в целом, который фиксируется в памяти в виде целостного мыслительного преобразования. Однако этот результат не имеет своего языкового выражения. Внешнее языковое выражение имеет конечный результат этого процесса – реферат, который непосредственно соотносится с исходным текстом. Это не просто результат редукции исходного текста, а результат его смыслового преобразования, обязательным этапом которого является смысловое свертывание, совершающееся в процессе осмысления и понимания исходного текста в целом. Смысловое преобразование является необходимым условием адекватности реферата тексту оригинала. Таким образом, реферативный перевод представляет собой особый вид речемыслительной деятельности, в которой ни перевод, ни реферирование не существуют отдельно. Операции перевода в ней тесно связаны с мыслительными операциями, которые обеспечивают свертывание текста. При этом свертывание является доминирующим процессом, поскольку эти операции начинают функционировать на первых же этапах осмысления языковых выражений [1].

На сегодняшний день существует несколько методов реферирования, которые, разделены на две большие группы. Первую составляют методы, основанные на анкетировании. К ним относятся фасетное и поаспектное реферирование. Вторая группа представляет собой методы, в основе которых лежит анализ смысловой структуры входного текста.

Из методов первой группы наиболее известно поаспектное реферирование. Данный метод предполагает семантический анализ текста с помощью его разбиения на отдельные содержательные аспекты. А именно, проблему и тему исследования, область знания, цель работы и полученные результаты, методы и условия проведенного исследования.

Другой метод реферирования был предложен А. И. Жолковой, которая обозначила его как фасетное реферирование. Он представляет собой анализ содержания конкретной сферы научной деятельности. Этот анализ

заключается в накладывании на текст сетки фасетов, соответствующей числу категорий, которые отражают специфику данной области в различных аспектах. Основным понятием этой методики считаются элементарные логические сообщения, которые создают реферат.

Представляется, что вышеназванные методы реферирования можно рассматривать как анкетирование: референту предлагается «анкета» с «вопросами», которые помогут ему найти в тексте исходного текста соответствующую информацию. Главным достоинством анкетных видов реферирования является ограничение заранее заданной схемой, которая отражает структуру предметной области, потребности специалистов/ Основным недостатком этих методов является обращение к интуиции референта, который должен выделить существенную информацию, а также детальность методики, что может значительно усложнить процесс реферирования. Можно предположить, что методы, основанные на анкетировании, ориентированы на создание специальных рефератов, поскольку в них отражается та информация, которая удовлетворяет потребности специалистов выбранной предметной области.

Вторую группу методов реферирования составляют методы выделения содержания входного текста на основе анализа реферируемого текста. К этой группе относится метод поаспектного реферирования, а именно разделение информации по степени значимости и полезности для потребителей. Предложенный метод реферирования опирается на содержательный и логический анализ первоисточника, который позволяет оценивать важность и приоритетность одних элементов информации относительно других, что делает методику достаточно универсальной. В связи с этим данный подход стал широко применяться в информационных центрах, и достаточно популярен в настоящее время [2].

1.2 Денотативная модель реферативного перевода текста

В данной работе содержание понимается как мыслительное преобразование, формирующееся в результате восприятия и понимания

конкретного текста и соответствующее некоторой динамической модели определенного фрагмента действительности. Это означает, что единицы содержания не тождественны единицам языка. Им соответствуют конкретные и целостные представления о том, что стоит за тем или иным языковым выражением. Все это дает основание назвать такие единицы содержания текста денотатами, а соответствующие им языковые выражения – именами денотатов. Тогда, структуру текста, содержащую такие языковые единицы можно назвать денотатной структурой [1].

Механизмом свертывания текста определим денотативную модель, основанную на методике денотативного анализа текста, которая рассматривается как инструмент для изучения внутренней формы речевого сообщения и для формализованного выделения содержания текста. Метод денотативного анализа текста обеспечивает возможность выразить содержание речевого сообщения в виде денотатного графа. Денотатный граф можно рассматривать как «перекодирование линейного текста в целостную, иконическую схему». Графовое представление материала вызывает полную перестройку текста: линейный текст преобразуется в иерархическую структуру, которая является моделью ситуации и отражает логику предметов и их отношений. Денотатный граф позволяет визуально продемонстрировать тематические уровни исходного текста – основную тему, подтемы и субподтемы. При этом устраняется языковая избыточность, свойственная первичному тексту. Денотативный анализ текста позволяет представить реферативный перевод как поэтапный переход $Т_1 \rightarrow$ ДС $\rightarrow Т_2$: сначала происходит переход от внешней языковой структуры текста к структуре его содержания, то есть его денотатной структуре, и затем переход от денотатной структуры к новой внешней форме вторичного текста. Следовательно, промежуточным звеном является денотатная структура документа, которая выступает в качестве результата понимания первичного документа и служит основой для создания реферативного текста. В данном переходе можно выделить два этапа, где первый ($Т_1 \rightarrow$ ДС) представляет

собой смысловую свертку материала, это есть этап понимания. Второй этап (ДС → $Т_2$) включает развертывание денотатной структуры в семантически адекватный первичному вторичный текст [3].

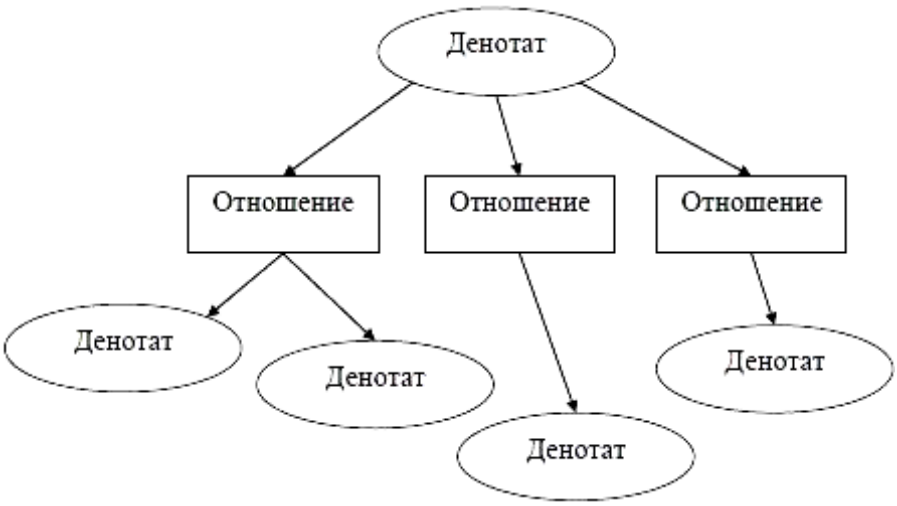

Рисунок 1 – Модель денотатного графа [3]

1.3 Понятие колокаций

Колокации понимаются как сочетание двух или более лексических единиц, характерное как для языка в целом, так и для определенного типа текстов. Поскольку текст это структурированная последовательность единиц разных уровней, колокации выступают важным объектом при исследовании процедур анализа текста. С их помощью создается возможность исследования структурных единиц текста разных текстовых уровней и их роль в процедурах анализа. Результаты такого исследования могут быть востребованы специалистами в самых разных областях, в том числе в области автоматизированного реферирования текстов. Анализ колокаций, которые входят в состав научных статей, в наибольшей степени отражает специфику появления «особых» единиц, характеризующих специфику предметной области и тип текста. Чаще всего колокации рассматриваются тогда, когда референт пытается выделить фразеологизмы из текста, а так же разнообразные варианты текстовой реализации лексических функций [4].

1.4 Основные понятия теории графов. Матрица весов

Понятие «граф» возникло в XVIII веке, когда математик Леонард Эйлер попытался решить задачу о Кенигсбергских мостах. Суть задачи состоит в

следующем: необходимо совершить пешую прогулку по Кёнигсбергу таким образом, что бы пройти по каждому мосту ровно один раз и вернуться в то же место, с которого началась прогулка. Для того, что бы решить эту задачу, Эйлер изобразил город в виде графа, в котором вершинами обозначил части города, а ребрами – мосты. Так, он доказал, что искомого маршрута для пешей прогулки не существует. Решение задачи Эйлера стало началом теории графов. В настоящее время теория графов является полноправной математической дисциплиной и включает в себя некоторые базовые понятия:

Неориентированным графом называется совокупность двух множеств: множества вершин и множества ребер.

Следующее, интересующее нас понятие – маршрут. Маршрутом в графе называют последовательность вершин и рёбер. Маршрут, в котором все вершины различны, называется простой цепью.

Взвешенный граф – граф, в котором каждому ребру поставлено в соответствии некоторое число – вес. В дальнейшем, понятие «взвешенный граф» еще не раз будет использоваться в работе. Именно такая структура легла в основу модели данных и эталонного графа в создаваемой автоматизированной системе.

В задачах, решаемых на ЭВМ графы удобно представлять в виде матриц. В данном случае использована матрица весов. Матрица весов – квадратная матрица, размерности $\boldsymbol{pp}$, где $\boldsymbol{p}$ – количество вершин. Элемент, стоящий в $\boldsymbol{i}$-том столбце и $\boldsymbol{j}$-той строке определяется по правилу: [5]

$$\boldsymbol{\alpha_{ij}} = \begin{Bmatrix} 0, \text{если вершины } \boldsymbol{v_i} \text{ и } \boldsymbol{v_j} \text{ совпадают} \\ \infty, \text{если вершины } \boldsymbol{v_i} \text{ и } \boldsymbol{v_j} \text{ не смежны} \\ \boldsymbol{w}, \text{если вершины } \boldsymbol{v_i} \text{ и } \boldsymbol{v_j} \text{ смежны, и } \boldsymbol{w} - \textit{вес ребра } (\boldsymbol{v_i}, \boldsymbol{v_j}) \end{Bmatrix} \quad (1)$$

1.5 Организация хранения графов в ЭВМ

Для создания адекватной системы автоматизированного реферирования необходимо выбрать такой способ организации хранения данных, при котором полученный входе анализа текста данные не будут искажаться и в

конечном результате сложатся в адекватный автореферат. Рассмотрим некоторые из способов хранения графов.

1.5.1 Матрица смежности

Один из способов хранения графов – матрица смежности. Она представляет собой двумерный массив, состоящий из значений весов дуг. Если дуга между вершинами отсутствует, то её вес принимают за 0. Если граф взвешенный, то в случае, когда дуга между двумя вершинами имеется, вписывается вес этой дуги. Если же граф не взвешенный, то матрица смежности состоит только из 0 и 1. Понятие «матрица смежности» применима только к ориентированным графам, именно такой граф используется для построения модели данных для создаваемой системы. Однако, как можно заметить, в матрице смежности часто нужно хранить большое количество нулей. Например, в предполагаемой матрице заложены всего 6 из 36 ячеек могут быть заполнены отличными от 0 цифрами. А значит, что всего 20% информации, хранимой в матрице, не представляют интерес для программиста. Кроме большого объема занимаемой памяти, матрица смежности имеет еще один существенный недостаток. Иногда в задачах требуется вводить не номер вершины, а номер дуги. Хранить такие номера матрица смежности не умеет, поэтому нужно реализовывать хранение дуг как-то иначе, что совсем неудобно.

1.5.2 Описание Бержа

Для того, что бы ускорить работу матрицы смежности на больших графах, можно использовать другой тип хранения графов – описание Бержа. Для каждой вершины хранится список смежных вершин. Чаще всего, для этого используется двумерный массив размерностью V^2 , в строке u которого хранятся номера вершин, смежных с u. В нулевой элемент строки u вписывается количество смежных вершин, хранящихся в данной строке. Вес дуг никак не хранится, а значит, для его хранения придется заводить еще один массив эквивалентной размерности, что требует дополнительной памяти.

1.5.3 Список дуг

Следующий способ хранения графа в памяти ЭВМ – список дуг. Чаще всего это массив размерности 3****E***. В первой строке массива хранится информация, из какой вершины начинается дуга, во второй – в какой кончается, а в третьей – вес дуги. Благодаря такому способу хранения почти вся таблица заполняется отличными от нуля значениями, что существенно сокращает объем используемой памяти. Такой способ хранения графа выгоден, если нужно узнать какую-либо информацию об ***i***-той дуге.

1.5.4 Список смежности

Для организации такого типа хранения графов, как список смежности обычно создается два массива: ***vert*** и ***adj***. Из каждой вершины выходит 0, 1 или более дуг. Если из вершины не выходит ни одной дуги, то в соответствующей ячейке массива ***vert*** будет храниться значение «0». Если из вершины выходит хотя бы одна дуга, то массив ***vert*** хранит номер ячейки массива ***adj***, где хранится конечная вершина дуги. Более того, в массиве ***adj*** хранится вес дуги и указатель на «конечную» вершину. Это самый удобный, производительный и несложный способ хранения графов в памяти компьютера [6].

1.6 Обзор имеющихся систем автоматического реферирования

1.6.1 Smart Search System (3S)

3S – это персональная поисковая система, работающая с использованием ассоциаций между терминами, выявленных при структуризации коллекции документов. Так же, программа предусматривает возможность расширения поиска за счет добавления в запрос связанных с реализуемым запросом терминов, которые уже содержатся в семантической цепи данной коллекции документов [7].

1.6.2 Inxight Summarizer (IS)

Программа *IS* – одна из немногих, традиционных программ реферирования. Она работает на основе выделения наиболее весомых

предложений из текста, используя синтаксические алгоритмы, или слова-подсказки [8].

1.6.3 TextAnalyst

Программа *TextAnalyst* работает только с русскоязычными текстами, выделяя именные группы и строя на из основе семантическую сеть – структуру взаимосвязанностей между именными группами [9].

1.6.4 Золотой ключик

Золотой ключик – это программная библиотека, работающая по принципу фильтрации на базе тезауруса. В качестве входных данных в программу загружается произвольный текст на русском языке, а на выходе формируется аннотация данного текста и список предметных областей, к которым можно отнести данный текст. В качестве аннотации используются предложения из входного текста, наиболее полно отражающие тематику [10].

1.6.5 МЛ Аннотатор

Программа МЛ Аннотатор составляет связный реферат документа. «Коэффициент сжатия» задаётся пользователем. Программа имеет два режима работы: собственно реферирование и выделение ключевых слов. В режиме реферирования из текста отбираются предложения, в наибольшей степени характеризующие его содержание. В режиме выделения ключевых слов производится выборка из текста наиболее информативных слов. Программа выделяет в тексте значимые слова, самостоятельные и зависимые предложения, определяет семантический вес предложений и удаляет незначащие фрагменты. Отобранные предложения при необходимости слегка перефразируются [11].

1.6.6 Особенности создаваемой системы «Denotator»

Стоит признать, что методы автоматизированного составления рефератов, которые используются в настоящее время, не удовлетворяют требованиям реферирования и качество вторичных текстов остаётся довольно низким. Это обусловлено тем, что при создании автореферата не учитываются связи и отношения между сущностями. Это дает стимул для

создания такой системы, которая будет составлять автореферат на основе исследования денотативной структуры текста, опираясь на вес связей между сущностями и целевую аудиторию.

2 ВЫБОР ПРОГРАММНЫХ СРЕДСТВ

2.1 Вспомогательные программы для исследования предметной области

2.1.1 Текстовый редактор Atom

Текстовый редактор *Atom* – это программа для редактирования текста и программного кода с удобным пользовательским интерфейсом. Она работает с большинством распространенных кодировок, а также включает в себя функцию подсветки синтаксиса. *Atom* можно использовать в качестве файлового менеджера. Его функции позволяют легко переключаться между файлами и папками, перемещать текстовые документы и просматривать структуру директорий. *Atom* открывает новые файлы в отдельных вкладках, так же как и большинство современных текстовых редакторов. Благодаря чему можно легко копировать и перемещать информацию. Встроена функция запоминания открытых вкладок при закрытии. По умолчанию в редакторе используется оформления в темных тонах, что удобно для работы с текстом в ночное время. *Atom* – бесплатная программа. Редактор является превосходным инструментом для разработчиков [12].

2.1.2 Пакет утилит Graphviz

Визуализация графа это способ представления структурной информации в виде диаграмм и абстрактных графов. Это имеет важное значение в области разработки программного обеспечения, баз данных и web-дизайна, машинного обучение, так и в визуальных интерфейсах для других технических областей.

Graphviz – программа с открытым исходным кодом это программное обеспечение для визуализации графов. Она имеет несколько основных наборов инструментов-программ, web и интерактивные графические интерфейсы, и вспомогательные инструменты, библиотеки и языковые привязки. *Graphviz* имеет много полезных функций для конкретных диаграмм, такие как выбор вариантов цвета, шрифтов, табличные макеты узлов, стили линий, гиперссылки. На практике, графики, как правило,

генерируются от внешних источников данных, но они также могут быть созданы и изменены вручную, либо как текстовые или графические файлы [13].

2.1.3 Программа MyStem

Эта программа умеет делать морфологический анализ русскоязычного теста и возвращать лингвистические характеристики слов, не входящих в словарь.

2.2 Выбор среды и языка программирования

2.2.1 Операционная система Ubuntu

Ubuntu – это основанная на *Debian* операционная система семейства *GNU/Linux*. Основная задача, которую выполняет *Ubuntu* — предоставление простого и удобного интерфейса для работы с компьютером, реализуя мощь, заложенную в *Linux*. ОС *Ubuntu* проста для освоения и использования. Простой и понятный интерфейс легко понимается людьми, а менеджер обновлений самостоятельно следит за актуальностью приложений и обновляет их по мере необходимости. Таким образом, снимается обязанность с пользователя самостоятельно обновлять приложения. В *Ubuntu* приложения могут быть установлены только пользователем с высоким уровнем привилегий. Таким образом, можно защитить на предприятиях компьютеры от вредоносных программ. Ещё одно правило, помимо разделения прав на установку ПО — устанавливать программы только из Центра приложений *Ubuntu*. Там находятся проверенные приложения, и сообщество прикладывает большие усилия, чтобы программы, размещённые там, были надёжны. Следуя данным рекомендациям, вы не подцепите на свой компьютер ни одного вируса.

Ubuntu всегда будет распространяться бесплатно, включая корпоративные версии и обновления безопасности. Для *Ubuntu* доступна полная коммерческая поддержка от *Canonical Ltd*. И сотен компаний по всему миру. *Ubuntu* включает наилучшие переводы и средства доступности для людей с ограниченными возможностями, которые только существуют в

виде открытого ПО. Диски с *Ubuntu* содержат только свободное программное обеспечение [14].

2.2.2 Язык программирования Python

Python — это свободный, интерпретируемый, объектно-ориентированный, расширяемый, встраиваемый язык программирования очень высокого уровня. Язык программирования *Python* имеет ряд преимуществ:

– «свободный» язык — все исходные тексты интерпретатора и библиотек доступны для любого, включая коммерческое, использование;

– интерпретируемый язык — потому что использует «позднее связывание»;

– объектно-ориентированный язык — классическая ОО модель, включая множественное наследование;

– расширяемый язык — имеет строго определенные *API* для создания модулей, типов и классов на языках *C* или *C++*;

– встраиваемый язык — имеет строго определенные *API* для встраивания интерпретатора в другие программы;

– язык очень высокого уровня — динамическая типизация, встроенные типы данных высокого уровня, классы, модули, механизм исключений.

Python — универсальный язык. Он широко используется во всем мире для самых разных целей — базы данных и обработка текстов, встраивание интерпретатора в игры, программирование *GUI* и быстрое создание прототипов (*RAD*). Кроме того, *Python* используется для программирования *Internet* и *Web* приложений — серверных (*CGI*), клиентских (роботы), *Web*-серверов и серверов приложений. *Python* обладает богатой стандартной библиотекой, и еще более богатым набором модулей, написанных третьими лицами. *Python* и приложения, написанные на нем, используют самые известные и крупные фирмы — *IBM, Yahoo!, Google.com, Hewlett Packard, Infoseek, NASA, Red Hat, CBS MarketWatch, Microsoft* [15].

2.2.3 Фреймворк для web-приложений Django

Django – это описание данных, которые хранятся в базе данных, выполненное в виде кода на языке *Python*. *Django* использует модель для фонового выполнения *SQL* и возвращает удобные структуры *Python* с данными, представляющими записи в таблицах вашей базы данных. *Django* также использует модели для представления высокоуровневых концепций, которые *SQL* вряд ли сможет обработать.

Django действует следующим образом:

– Самодиагностика занимает ресурсы и несовершенна. Для того, чтобы предоставить удобный *API* для доступа к данным, *Django* должна как-то получить информацию о содержимом базы данных и существует два способа сделать это. Первый: можно явно описать данные в *Python*. Второй: самодиагностика базы данных при работе приложения для определения моделей данных. Второй способ кажется более очевидным, так как метаданные в ваших таблицах хранятся в одном месте, но он приводит к нескольким проблемам.

– Разрабатывать с помощью *Python* удобно и хранение всего в виде его кода сокращает число«переключений контекста», которые приходится делать вашему мозгу. Если придерживаться единой среды разработки менталитета, это повысит продуктивность работы. Необходимость писать *SQL*, затем код *Python*, а потом опять *SQL*, отрицательно сказывается на производительности разработчика.

– Когда модели данных хранятся в виде кода, а не в базе данных, это упрощает управление версиями этих моделей. Это поможет вам хранить историю всех изменений моделей.

– *SQL* позволяет лишь определённый уровень хранения метаинформации о формате данных. Большинство систем управления базами данных не предоставляют специализированных типов данных для представления адресов электронной почты или *URL*. Модели *Django* предоставляют. Преимущество высокоуровневых типов данных в улучшении продуктивности и в большем повторном использовании кода.

– *SQL* несовместим между разными платформами баз данных. При распространении *web*-приложения, более практично передавать модуль на языке *Python*, который описывает формат ваших данных, вместо отдельных наборов операторов.

Недостаток такого подхода, тем не менее, в том, что есть возможность для кода *Python* выйти из синхронизации с содержимым базы данных. При внесении изменений в модель, вам потребуется также изменить содержимое базы данных. Мы рассмотрим некоторые стратегии для решения этой проблемы далее в этой главе.

Django включает в себя утилиту, которая может генерировать модели по метаинформации существующей базы данных. Это полезно для быстрого получения и начала работы с уже существующей информацией [16].

2.3 Инструменты Python

2.3.1 Модуль SQLite для Python

SQLite – это встраиваемая в процесс, реализация SQL-машины. Этот модуль является полностью открытым и свободным. Используется многими операционными системами, такими как Symbian и Android и встроен в языки программированная Python и PHP. Есть мнение, что это самая распространенная в мире СУБД. С помощью этого модуля, представляется возможным, не выходя из Python создавать таблицы и запросы, с помощью простых операторов SQLite.

2.3.2 Beautiful Soup

Beautiful Soup – это парсер для синтаксического разбора файлов *HTML/XML*, написанный на языке программирования *Python*, который может преобразовать даже неправильную разметку в дерево синтаксического разбора. Для работы конструктору *Beautiful Soup* требуется документ *XML* или *HTML* в виде строки (или открытого файлоподобного объекта). Он произведет синтаксический разбор и создаст в памяти структуры данных, соответствующие документу. Если обработать с помощью *Beautiful Soup* хорошо оформленный документ, то разобранная структура будет выглядеть

так же, как и исходный документ. Но если его разметка будет содержать ошибки, то *Beautiful Soup* использует эвристические методы для построения наиболее подходящей структуры данных. *Beautiful Soup* вычисляет наиболее вероятные места для закрывающих тегов, даже если они отсутствуют в исходном документе. Одним из общеизвестных недостатков *Beautiful Stone Soup* является то, что он не знает о самозакрывающихся тегах. В *HTML* имеется фиксированный набор самозакрывающихся тегов, но в случае с *XML* все зависит от того, что записано в *DTD*. Вы можете сообщить *Beautiful Stone Soup*, что определенные теги являются самозакрывающимися, передав их имена через аргумент конструктора *self Closing Tags* [17].

2.3.3 Библиотека *rutermextract*

Rutermextract – это библиотека для извлечения ключевых слов из текстов на русском языке. Является аналогом библиотеки topia.termextract, которая делает то же самое с англоязычными текстами. Извлеченные ключевые слова приводятся в нормальную форму и упорядочиваются от более важных к менее важным.

Возможности *Rutermextract*:

- автоматическое присвоение тегов;
- вычисление похожести текстов на основе извлеченных ключевых слов.

Библиотека извлекает ключевые слова на основе заранее заданных правил. На данный момент это единственный возможный вариант, поскольку для русского языка не существует открытого синтаксического корпуса, который можно использовать для обучения синтаксических моделей.

Основные проблемы:

1. Неполные правила. Например, сейчас не извлекаются ключевые слова, содержащие предлоги («вор в законе», «сосед по парте»). Эта проблема может решаться при дальнейшем развитии библиотеки, но правил, которые могли бы охватить все случаи быть не может.

2. Неоднозначность при морфологическом разборе. Сейчас она разрешается выбором наиболее вероятного варианта, что в некоторых случаях неверно. Проблема может проявляться как при извлечении ключевых слов, так и при их нормализации. (Например, из заголовка «Мальчика назвали в честь нападающего футбольного клуба» будет извлечена фраза «нападающий футбольный клуб».)

3. Ложные ключевые слова. Некоторые извлеченные фразы могут не являться на самом деле ключевыми. Размера текста не всегда бывает достаточно для того, чтобы отличить важные для текста слова от неважных, основываясь только на количестве употреблений. Поэтому необходимо использовать сторонние модели (например, *tf-idf*) для определения важности ключевых слов [18].

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПОИСКА ДЕНОТАТОВ

3.1 Создание модели данных автоматизированной системы «Denotator»

В общем случае, модель данных – это средство абстракции, которое позволяет отделить факты от их интерпретации в программе и одновременно обеспечить возможность представления соотношения данных.

Для создания адекватной автоматизированной системы, в первую очередь необходимо определить, какие классы должна содержать модель данных. Как описано выше, модель данных будем строить в виде денотатного графа, а это значит, что у каждой связи должен быть определённый вес и значение. Для того что бы полученный на выходе реферат был синтаксически и семантически корректным необходимо выделить то количество сущностей, с помощью которого станет возможным описать научно-технический текст. Сложность состоит в том, что все тексты имеют разную структуру и размер. Так же, в научно-технической литературе встречаются как публицистические, так и специальные тексты, которые отличаются не только лексическими оборотами и синтаксисом, но и структурой и, что более важно, скопосом, на который направлен данный текст. Итак, в ходе построения модели данных необходимо решить несколько задач. Во-первых, она не должна быть узко-ориентированной: все сущности и связи модели в первую очередь должны обладать гибкостью и расширяемостью. Во-вторых, в модели данных необходимо предусмотреть изменение семантики и скопоса и реализовать связи между всеми сущностями в ней.

В ходе разработки модели данных было выделено 16 сущностей, а именно: скопос, словосочетание, денотат, отношение, ключевое слово, проблематика текста, вопрос, предметная область, тема исследования, словарь, слово, текст, устоявшиеся речевые обороты, член предложения, аргументы, предложение. Все эти сущности соединены между собой дугами,

каждой из которых присвоено «значение связи». Построение модели реализовано в программном пакете Graphviz, код представлен на рисунке 1.

```
digraph ex01 {
        rankdir=LR;
        size="8,5"
        node [shape = box];
        "Скопос" -> "Аргументы" [ label = "Характеризуется", style=bold ];
        "Словосочетание" -> "Аргументы" [ label = "Состоят из", style=bold ];
        "Денотат" -> "Отношение" [ label = "О", style=bold ];
        "Денотат" -> "Словарь" [ label = "Активирует", style=bold ];
        "Денотат" -> "Предметная область" [ label = "Принадлежит", style=bold ];
        "Словосочетание" -> "Денотат" [ label = "Обозначает", style=bold ];
        "Отношение" -> "Денотат" [ label = "О", style=bold ];
        "Словосочетание" -> "Отношение" [ label = "Выражено", style=bold ];
        "Скопос" -> "Предметная область" [ label = "Определяется", style=bold ];
        "Ключевое слово" -> "Скопос" [ label = "Определяется", style=bold ];
        "Проблематика текста" -> "Вопрос" [ label = "Определяется", style=bold ];
        "Словосочетание" -> "Вопрос" [ label = "Входит", style=bold ];
        "Вопрос" -> "Ключевое слово" [ label = "Состоит из", style=bold ];
        "Проблематика текста" -> "Ключевое слово" [ label = "Состоит из", style=bold ];
        "Предметная область" -> "Проблематика текста" [ label = "Определяется", style=bold ];
        "Тема исследования" -> "Проблематика текста" [ label = "Состоит из", style=bold ];
        "Словосочетание" -> "Словарь" [ label = "Берётся из", style=bold ];
        "Словарь" -> "Устоявшиеся речевые обороты" [ label = "Содержит", style=bold ];
        "Словосочетание" -> "Предметная область" [ label = "Характеризуется", style=bold ];
        "Слово" -> "Словосочетание" [ label = "Входит в", style=bold ];
        "Словосочетание" -> "Член предложения" [ label = "Выражено", style=bold ];
        "Словосочетание" -> "Текст" [ label = "Входит", style=bold ];
        "Текст" -> "Предметная область" [ label = "Принадлежит", style=bold ];
        "Слово" -> "Предметная область" [ label = "Принадлежит", style=bold ];
        "Предметная область" -> "Предметная область" [ label = "Входит в", style=bold ];
        "Предметная область" -> "Денотат" [ label = "Деактивирует", style=bold ];
        "Ключевое слово" -> "Тема исследования" [ label = "Определяется", style=bold ];
        "Устоявшиеся речевые обороты" -> "Предложение" [ label = "Входят в", style=bold ];
        "Предложение" -> "Текст" [ label = "Входит в", style=bold ];
        "Член предложения" -> "Предложение" [ label = "Входит в", style=bold ];
}
```

Рисунок 1 – Код графа для построения модели данных

После компиляции кода на выходе программы получаем модель данных в виде денотатного графа. Фрагмент модели данных представлена на рисунке 2, полную схему можно посмотреть в Приложении В.

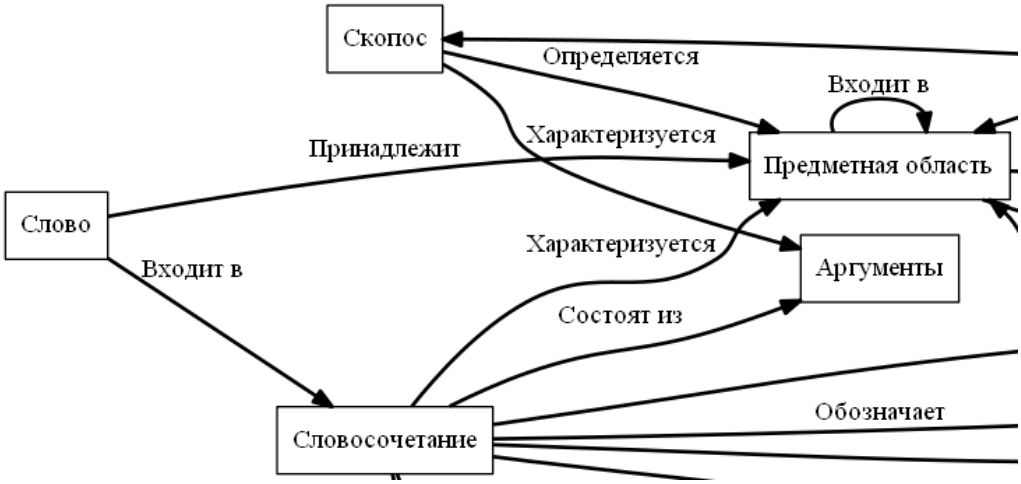

Рисунок 2 – Фрагмент модели данных

3.2 Создание эталонного графа (ЭГ)

Построению денотатного графа, предшествует денотативный анализ текста либо предметной области. Особенностью этого анализа является то, что его единицей является не отдельный фрагмент, а весь текст или вся область в целом. Графовый анализ текстов по заданной предметной области вызывает полное его преобразование, что объясняется неизолированностью его внешней и внутренней форм. Таким образом, в процессе уточнения понятий в денотатной структуры преодолеваются противоречия между целостностью мыслительного содержания и линейностью, дискретностью языкового способа его выражения, с одной стороны, и неизоморфностью единиц внутреннего и внешнего планов текста – с другой. Кроме того, при построении денотатного графа происходит и определенный отбор денотатов, устраняются те из них, которые не имеют непосредственного отношения к моделируемой ситуации. Другими словами, при переходе от линейной формы представления текста к графовой устраняется языковая и предметная избыточность, свойственная первичным текстам. Из этого следует, что полученный таким образом денотатный граф представляет собой свернутое отображение структуры содержания текста, которой могут соответствовать различные языковые формы.

Для создания системы автоматизированного поиска денотатов определили предметную область «Информатика». Она включает в себя пять подтем: теоретическая информатика, программирование, искусственный интеллект, защита информации и вычислительная техника. Таким образом ЭГ будет состоять из пяти связанных общей предметной областью денотатных графов.

Как и модель данных, создадим ЭГ в программном пакете Graphviz. На рисунках 3-8 представлены фрагменты эталонного графа. Листинг модели данных можно изучить в Приложении Г.

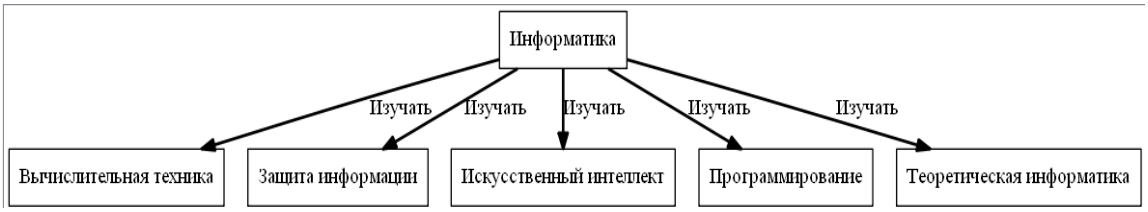

Рисунок 3 – Общая структура предметной области

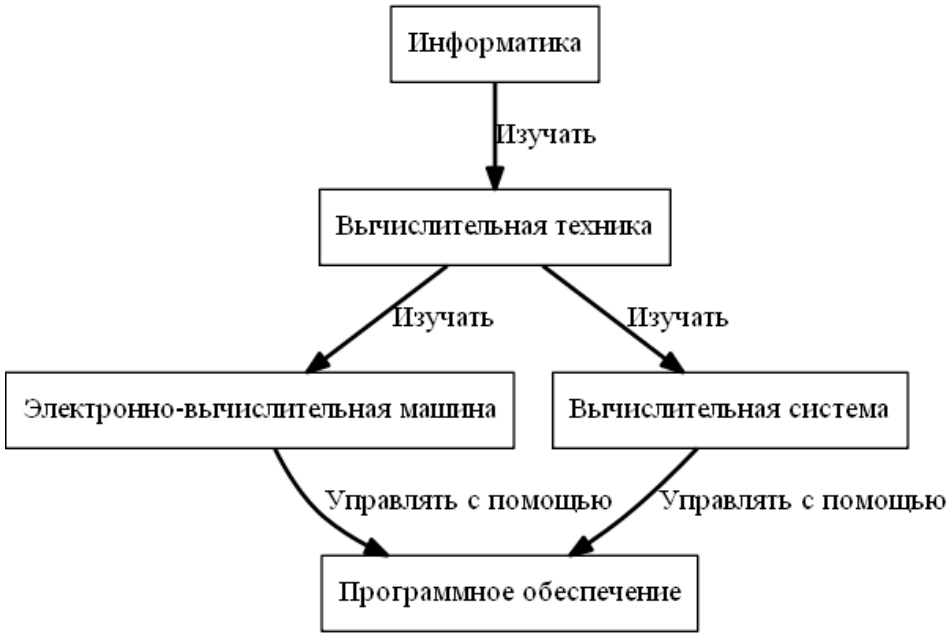

Рисунок 4 – Фрагмент предметной области в ЭГ «Вычислительная техника»

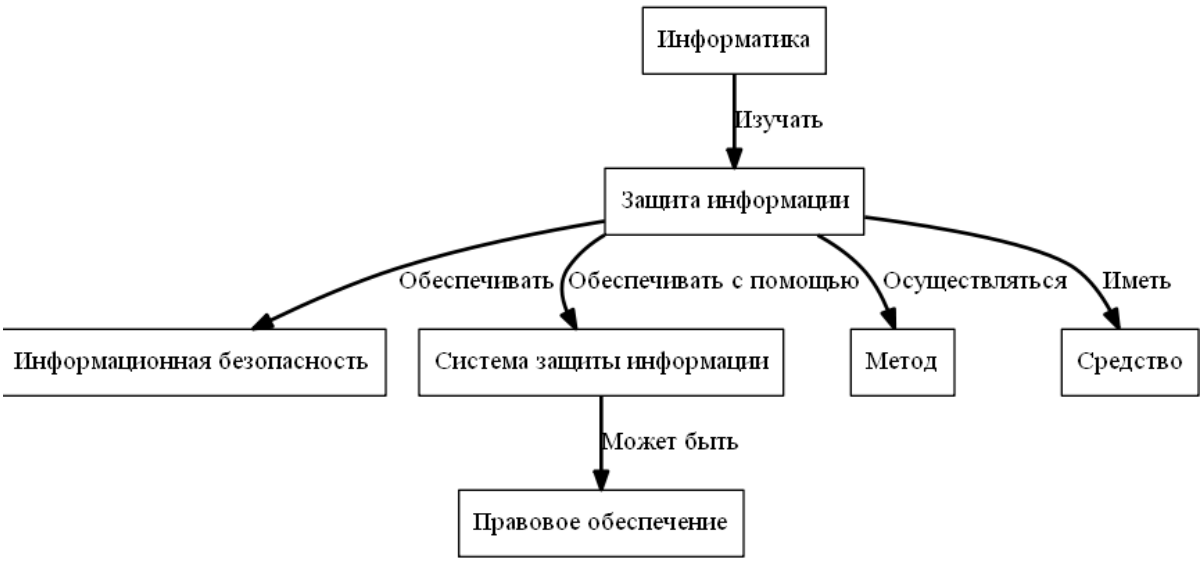

Рисунок 5 – Фрагмент предметной области в ЭГ «Защита информации»

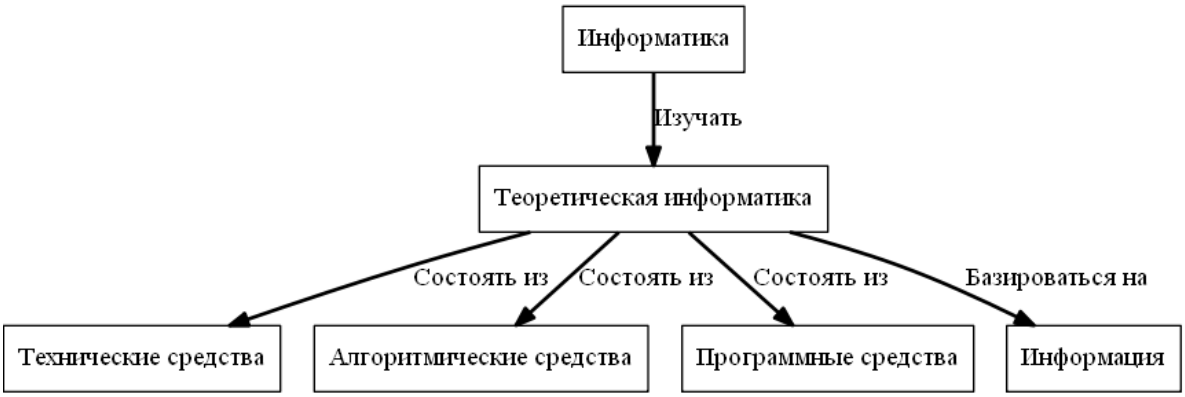

Рисунок 6 – Фрагмент предметной области в ЭГ «Теоретическая информация»

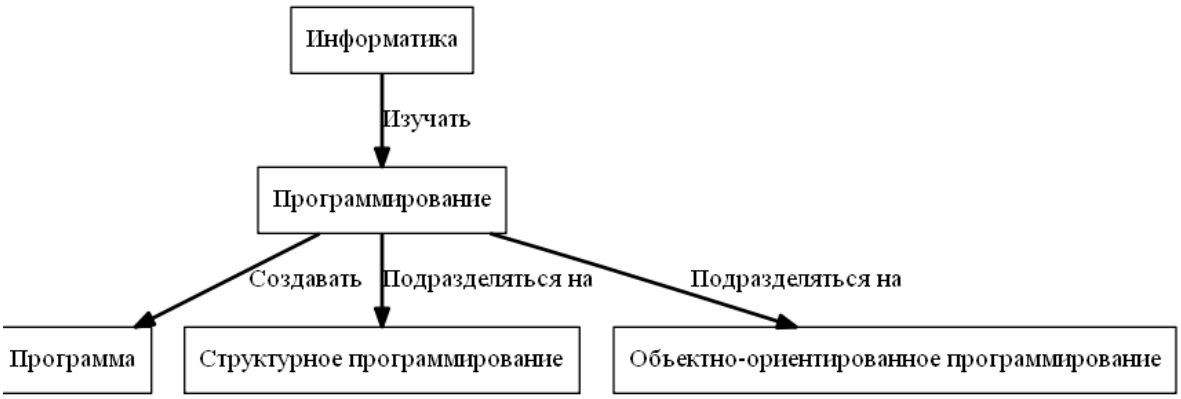

Рисунок 7 – Фрагмент предметной области в ЭГ «Программирование»

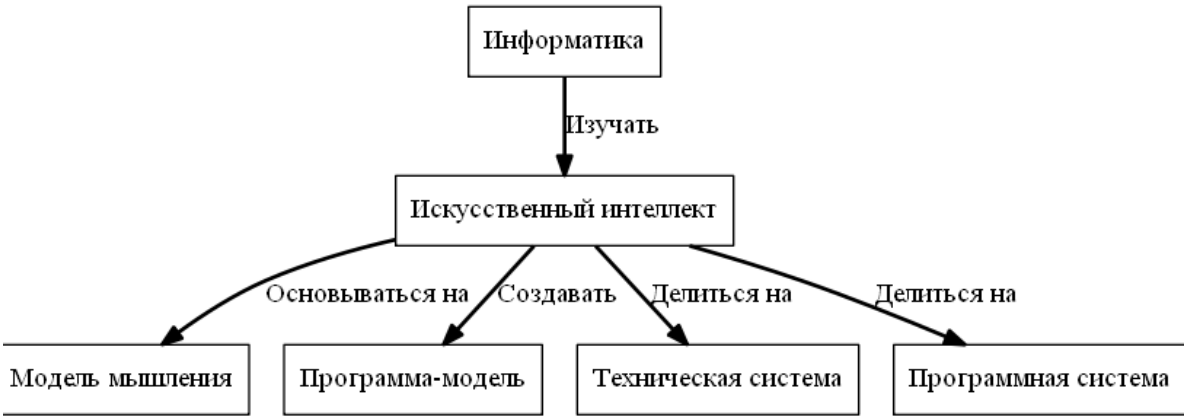

Рисунок 8 – Фрагмент предметной области в ЭГ «Искусственный интеллект»

3.3 Извлечение колокаций и проверка связей в ЭГ

Для проверки предположения об оптимальности извлеченных денотатов, написанного на основании графа, был проведен предварительный

эксперимент, который заключался в следующем. Было взято пять текстов (рисунки 9-13) по пяти тематикам предметной области соответственно.

```
>>> from rutermextract import TermExtractor
>>> term_extractor = TermExtractor()
>>> text = u'Каждый процессор способен выполнять вполне определенный набор универсальных
инструкций, называемых чаще всего машинными командами. Каков именно этот набор, определяе
тся устройством конкретного процессора, но он не очень велик и в основном аналогичен для
различных процессоров. Работа элнктронно-вычислительной машины состоит в выполнении после
довательности таких команд, подготовленных в виде программы. Процессор способен организов
ать считывание очередной команды, ее анализ и выполнение, а также при необходимости приня
ть данные или отправить результаты их обработки на требуемое устройство. Выбрать, какую и
нструкцию программы исполнять следующей, также должен сам процессор, причем результат это
го выбора часто может зависеть от обрабатываемой в данный момент информации.'
>>> for term in term_extractor(text):
...   print (term.normalized, term.count)
```

Рисунок 9 – Текст 1 для предметной области «Вычислительная техника»

```
>>> from rutermextract import TermExtractor
>>> term_extractor = TermExtractor()
>>> text = u'Криптография представляет собой совокупность методов преобразования данных,
направленных на то, чтобы сделать эти данные бесполезными для злоумышленника. Такие преоб
разования позволяют решить два главных вопроса, касающихся безопасности информации: защит
у конфиденциальности, защиту целостности.Проблемы защиты конфиденциальности и целостности
 информации тесно связаны между собой, поэтому методы решения одной из них часто применим
ы для решения другой.'
>>> for term in term_extractor(text):
...   print (term.normalized, term.count)
```

Рисунок 10 – Текст 2 для предметной области «Защита информации»

```
>>> from rutermextract import TermExtractor
>>> term_extractor = TermExtractor()
>>> text = u'Логический подход к созданию систем искусственного интеллекта основ
ан на моделировании рассуждений. Теоретической основой служит логика. Логический
 подход может быть проиллюстрирован применением для этих целей языка и системы л
огического программирования Пролог. Программы, записанные на языке Пролог, предс
тавляют наборы фактов и правил логического вывода без жесткого задания алгоритма
 как последовательности действий, приводящих к необходимому результату.'
>>> for term in term_extractor(text):
...   print (term.normalized, term.count)
```

Рисунок 11 – Текст 3 для предметной области «Искусственный интеллект»

```
>>> from rutermextract import TermExtractor
>>> term_extractor = TermExtractor()
>>> text = u'При создании средних по размеру приложений (несколько тысяч строк исходного
кода) используется структурное программирование, идея которого заключается в том, что стр
уктура программы должна отражать структуру решаемой задачи, чтобы алгоритм решения был яс
но виден из исходного текста. Для этого надо иметь средства для создания программы не тол
ько с помощью трех простых операторов, но и с помощью средств, более точно отражающих кон
кретную структуру алгоритма. С этой целью в программирование введено понятие подпрограммы
 - набора операторов, выполняющих нужное действие и не зависящих от других частей исходно
го кода.'
>>> for term in term_extractor(text):
...   print (term.normalized, term.count)
```

Рисунок 12 – Текст 4 для предметной области «Программирование»

```
>>> from rutermextract import TermExtractor
>>> term_extractor = TermExtractor()
>>> text = u'Современный компьютер может обрабатывать числовую, текстовую, графическую, з
вуковую и видео информацию. Все эти виды информации в компьютере представлены в двоичном
коде.Для этого используется алфавит мощностью всего два символа 0 и 1. Связано это с тем,
 что удобно представлять информацию в виде последовательности электрических импульсов: им
пульс отсутствует, импульс есть. Такое кодирование принято называть двоичным, а сами логи
ческие последовательности нулей и единиц - машинным языком.'
>>> for term in term_extractor(text):
...   print (term.normalized, term.count)
```

Рисунок 13 – Текст 5 для предметной области «Теоретическая информатика»

С помощью инструмента *rutermextract*, из выбранных тестов извлекли колокации (рисунки 14-18).

```
какая инструкция программы 1
данный момент информации 1
электронно-вычислительная машина 1
универсальные инструкции 1
требуемое устройство 1
такие команды 1
сам процессор 1
различные процессоры 1
очередной команда 1
определённый набор 1
называемые чаща 1
машинные команды 1
конкретное процессор 1
каждый процессор 1
выполнение последовательности 1
вид программы 1
устройство 1
считывание 1
результаты 1
результат 1
работа 1
процессор 1
основное 1
обработка 1
необходимость 1
набор 1
выполнение 1
выбор 1
велик 1
анализ 1
```

Рисунок 14 – Колокации предметной области «Вычислительная техника»

```
защита целостности.проблемы защиты конфиденциальности 1
совокупность методов преобразования 1
касающиеся безопасность информации 1
два главных вопроса 1
целостность информации 1
такие преобразование 1
методы решения 1
защита конфиденциальности 1
решение 1
криптография 1
злоумышленник 1
```

Рисунок 15 – Колокации предметной области «Защита информации»

```
логический подход 2
пролог 2
жёсткое задание алгоритма 1
цели языка 1
теоретическая основа 1
создание систем 1
последовательность действий 1
необходимый результат 1
наборы фактов 1
моделирование рассуждений 1
логический программирование 1
логический вывод 1
искусственное интеллект 1
язык 1
система 1
программа 1
применение 1
правила 1
логика 1
```

Рисунок 16 – Колокации предметной области «Искусственный интеллект»

```
исходный код 2
отражающие конкретная структура алгоритма 1
три простых операторов 1
выполняющие нужное действие 1
тысячи строк 1
структурное программирование 1
структура программы 1
создание программы 1
решаемая задача 1
размер приложений 1
понятие подпрограммы 1
помощь средств 1
исходный текст 1
другие части 1
алгоритм решения 1
цель 1
структура 1
средство 1
создание 1
программирование 1
помощь 1
операторы 1
набор 1
идея 1
```

Рисунок 17 – Колокации предметной области «Программирование»

```
информация 2
импульс 2
сами логические последовательность нулей 1
электрические импульсы 1
такое кодирование 1
современный компьютер 1
машинный язык 1
два символа 1
виды информации 1
вид последовательности 1
мощность 1
компьютер 1
единицы 1
видео 1
алфавит 1
```

Рисунок 18 – Колокации предметной области «Теоретическая информация»

После извлечения ключевых слов и на их основе построили для каждого текста денотатный граф на основе ЭГ. Затем с помощью анализа появлений колокаций в графе определили вес связей (рисунки 19-23).

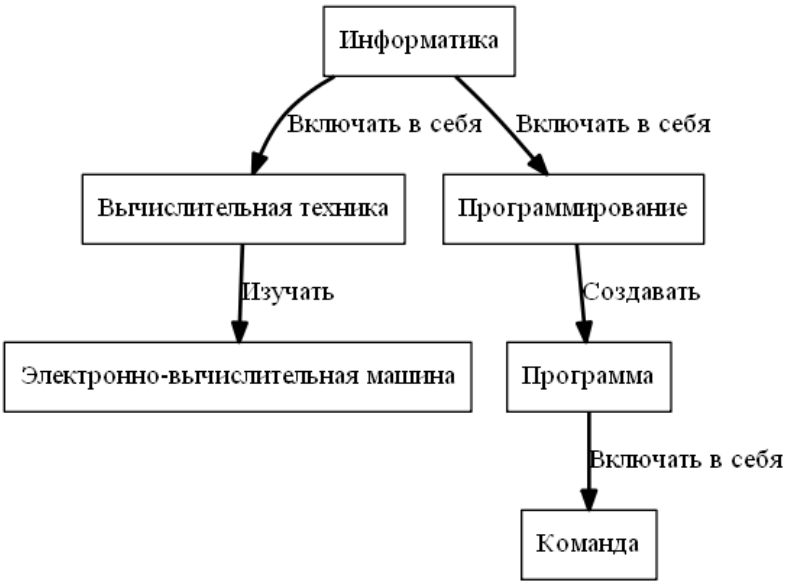

Рисунок 19 – Денотатный граф текста 1

Рисунок 20 – Денотатный граф текста 2

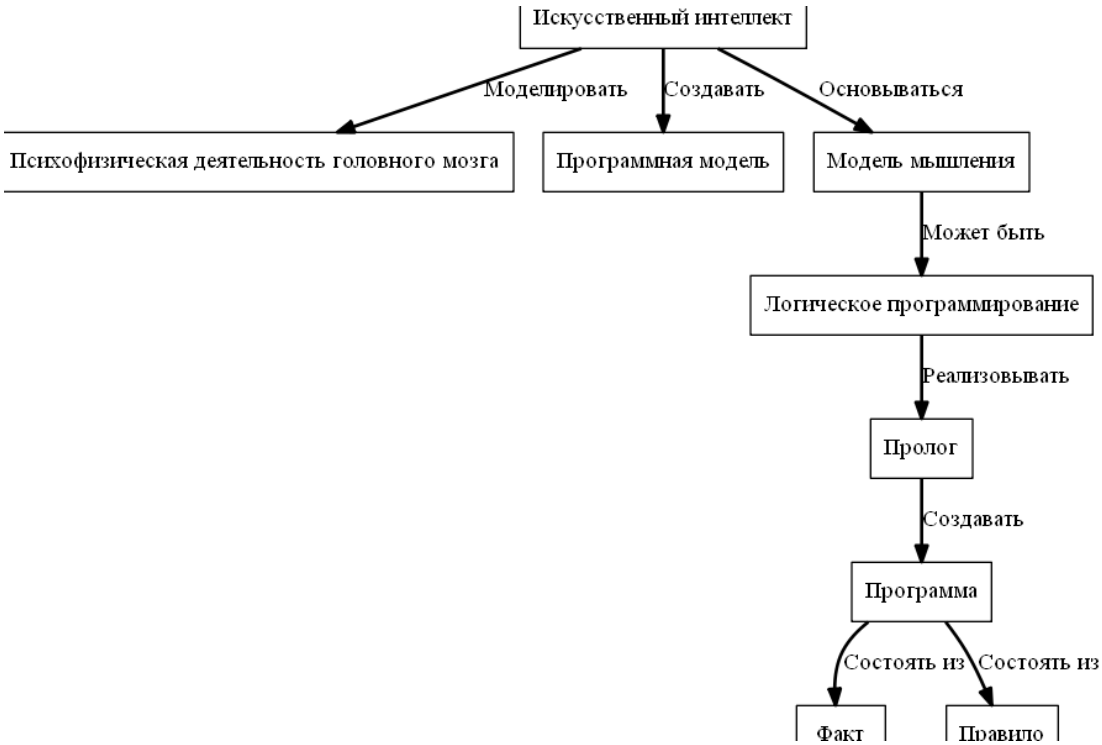

Рисунок 21 – Денотатный граф текста 3

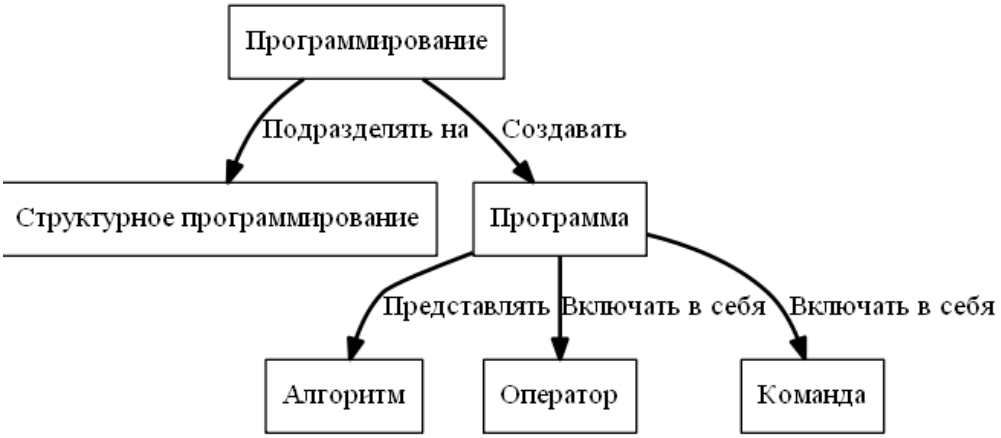

Рисунок 22 – Денотатный граф текста 4

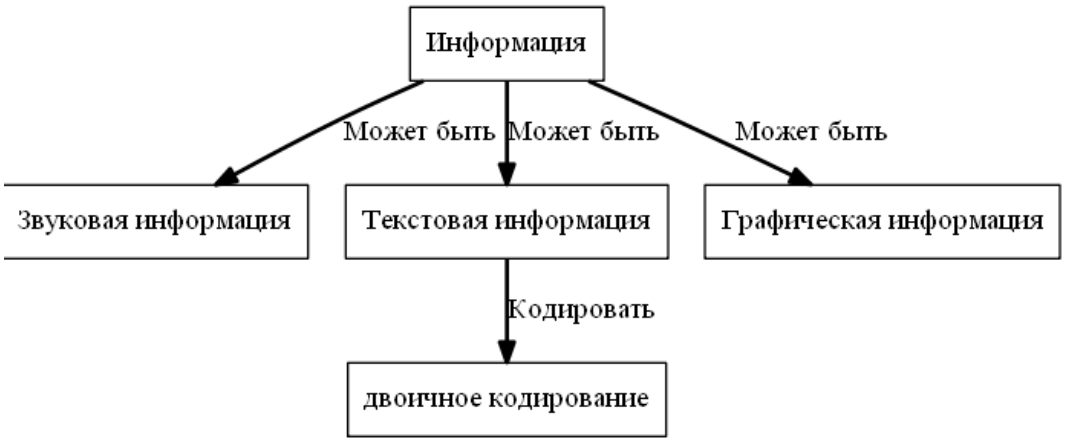

Рисунок 23 – Денотатный граф текста 5

3.4 Создание БД для автоматизированной системы «Denotator»

База данных для автоматизированной системы «Denotator» создана в *SQLite3*. Она содержит 16 классов, соответствующих модели данных. Между этим классам, существует связь, вес которой мы будем рассчитывать в ходе разработки непосредственно приложения. Перейдем в папку, где будет храниться проект и создадим его, так же приложение внутри проекта. Результаты представлены на рисунке 24.

```
student@student-HP-655-Notebook-PC:~/1$ django-admin.py startproject diplom
student@student-HP-655-Notebook-PC:~/1$ cd diplom
student@student-HP-655-Notebook-PC:~/1/diplom$ django-admin.py startapp denotat
student@student-HP-655-Notebook-PC:~/1/diplom$
```

Рисунок 24 – Создание проекта и приложения

В файле *settings.py* добавили название нашего приложения в категорию с перечисленными приложениями проекта. Результаты представлены на рисунке 25.

```
1  # Django settings for diplom project.
2
3  DEBUG = True
4  TEMPLATE_DEBUG = DEBUG
5
6  ADMINS = (
7      #('Your Name', 'your_email@example.com'),
8  )
9
10 MANAGERS = ADMINS
11
12 DATABASES = {
13     'default': {
14         'ENGINE': 'django.db.backends.sqlite3', # Add 'postgresql_psycopg2', 'mysql',
   'sqlite3' or 'oracle'.
15         'NAME': '/home/student/VKR/diplom/db.diplom',                      # Or path
   to database file if using sqlite3.
16         'USER': '',                      # Not used with sqlite3.
17         'PASSWORD': '',                  # Not used with sqlite3.
18         'HOST': '',                      # Set to empty string for localhost. Not used
   with sqlite3.
```

Рисунок 25 – Редактирование файла *settings.py*

Для создания таблицы и полей в файле *models.py* директории *denotat* с созданным приложением создадим класс *Poisk*. В файл должно входить описание 16-ти классов. Результаты представлены на рисунках 26 и 27.

```python
from django.db import models


class Argument(models.Model):

        Key_Collocation = models.CharField(max_length=255)

class Denotat(models.Model):

        Name = models.CharField(max_length=255)

class Attitude(models.Model):

        Direction = models.CharField(max_length=255)
        Probability = models.CharField(max_length=255)

class Scopos(models.Model):

        The_target_audience = models.CharField(max_length=255)

class Question(models.Model):

        User_request = models.CharField(max_length=255)

class Text_problematics(models.Model):

        Objectives = models.CharField(max_length=255)
        Tasks = models.CharField(max_length=255)

class Dictionary(models.Model):

        Notion = models.CharField(max_length=255)
        Deskription = models.CharField(max_length=255)
```

Рисунок 26 – Содержание файла *models.py*

```python
class Text(models.Model):

        Caption = models.CharField(max_length=255)
        Author = models.CharField(max_length=255)
        Output = models.CharField(max_length=255)

class Subject_area(models.Model):

        Name_area = models.CharField(max_length=255)
        Deskription_area = models.CharField(max_length=255)

class Key_Words(models.Model):

        Repetition_rate = models.CharField(max_length=255)
        Science_area = models.CharField(max_length=255)

class Figure_of_speech(models.Model):

        Linguistic_characteristics_of_speech = models.CharField(max_length=255)

class Sentence(models.Model):

        Linguistic_characteristics_sentence = models.CharField(max_length=255)


class Word(models.Model):

        Infinitive  = models.CharField(max_length=255)
        Linguistic_characteristics_word = models.CharField(max_length=255)

class Research_topic(models.Model):

        Section_science_area = models.CharField(max_length=255)
```

Рисунок 27 – Содержание файла *models.py*

Построим таблицу и, соответственно базу данных (рисунок 28-33), при этом программа выдаст запрос на создание суперпользователя для базы (рисунок 29). Мы согласимся и зададим имя и пароль. В файле *settings.py,* в

директории с приложением создаём файл *admin.py* и в файле urls прописываем соответствующий адрес (рисунок 34).

```
student@student-HP-655-Notebook-PC:~/1/diplom$ python manage.py syncdb
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table django_admin_log
Creating table denotat_argument
Creating table denotat_denotat
Creating table denotat_attitude
Creating table denotat_scopos
Creating table denotat_question
Creating table denotat_text_problematics
Creating table denotat_dictionary
Creating table denotat_collocation
Creating table denotat_text
Creating table denotat_subject_area
Creating table denotat_key_words
Creating table denotat_figure_of_speech
Creating table denotat_sentence
Creating table denotat_word
Creating table denotat_research_topic
```

Рисунок 28 – Создание базы данных

```
You just installed Django's auth system, which means you don't have any superuse
rs defined.
Would you like to create one now? (yes/no): y
Please enter either "yes" or "no": yes
Username (leave blank to use 'student'): kate
E-mail address: eriskina.katena@mail.ru
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

Рисунок 29 – Создание суперпользователя

```
student@student-HP-655-Notebook-PC:~/1/diplom$ python manage.py sqlall denotat
BEGIN;
CREATE TABLE "denotat_argument" (
    "id" integer NOT NULL PRIMARY KEY,
    "Key_Collocation" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_denotat" (
    "id" integer NOT NULL PRIMARY KEY,
    "Name" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_attitude" (
    "id" integer NOT NULL PRIMARY KEY,
    "Direction" varchar(255) NOT NULL,
    "Probability" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_scopos" (
    "id" integer NOT NULL PRIMARY KEY,
    "The_target_audience" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_question" (
    "id" integer NOT NULL PRIMARY KEY,
    "User_request" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_text_problematics" (
    "id" integer NOT NULL PRIMARY KEY,
    "Objectives" varchar(255) NOT NULL,
    "Tasks" varchar(255) NOT NULL
)
```

Рисунок 30 – Создание таблицы БД

```
CREATE TABLE "denotat_dictionary" (
    "id" integer NOT NULL PRIMARY KEY,
    "Notion" varchar(255) NOT NULL,
    "Deskription" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_collocation" (
    "id" integer NOT NULL PRIMARY KEY,
    "Linguistic_characteristics_collocation" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_text" (
    "id" integer NOT NULL PRIMARY KEY,
    "Caption" varchar(255) NOT NULL,
    "Author" varchar(255) NOT NULL,
    "Output" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_subject_area" (
    "id" integer NOT NULL PRIMARY KEY,
    "Name_area" varchar(255) NOT NULL,
    "Deskription_area" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_key_words" (
    "id" integer NOT NULL PRIMARY KEY,
    "Repetition_rate" varchar(255) NOT NULL,
    "Science_area" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_figure_of_speech" (
    "id" integer NOT NULL PRIMARY KEY,
    "Linguistic_characteristics_of_speech" varchar(255) NOT NULL
```

Рисунок 31 – Создание таблицы БД

```
CREATE TABLE "denotat_sentence" (
    "id" integer NOT NULL PRIMARY KEY,
    "Linguistic_characteristics_sentence" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_word" (
    "id" integer NOT NULL PRIMARY KEY,
    "Infinitive" varchar(255) NOT NULL,
    "Linguistic_characteristics_word" varchar(255) NOT NULL
)
;
CREATE TABLE "denotat_research_topic" (
    "id" integer NOT NULL PRIMARY KEY,
    "Section_science_area" varchar(255) NOT NULL
)
;
COMMIT;
student@student-HP-655-Notebook-PC:~/1/diplom$
```

Рисунок 32 – Создание таблицы БД

Рисунок 33 – Созданная БД

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
from django.conf.urls.defaults import *
from denotat.views import denotat
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'diplom.views.home', name='home'),
    # url(r'^diplom/', include('diplom.foo.urls')),

    # Uncomment the admin/doc line below to enable admin documentation:
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    url(r'^admin/', include(admin.site.urls)),
    url('denotat/$',denotat)
)
```

Рисунок 34 – Содержание файла *urls.py*

В файл *admin.py* запишем все классы, доступ к которым необходим для суперпользователя. Результаты представлены на рисунке 35.

```
from denotat.models import Argument, Denotat,  Attitude, Scopos, Question,
Text_problematics, Dictionary, Collocation, Text,  Subject_area, Key_Words,
Figure_of_speech, Sentence, Word,Research_topic
from django.contrib import admin

#from denotat.admin import custom_admin_site
admin.site.register (Argument, Denotat,  Attitude, Scopos, Question, Text_problematics,
Dictionary, Collocation, Text,  Subject_area, Key_Words, Figure_of_speech, Sentence,
Word,Research_topic)

#, site=custom_admin_site
```

Рисунок 35 – Содержимое файла *admin.py*

Запустим сервер (рисунок 36) и проверим на ошибки основной файл проекта *manage.py* (рисунок 37).

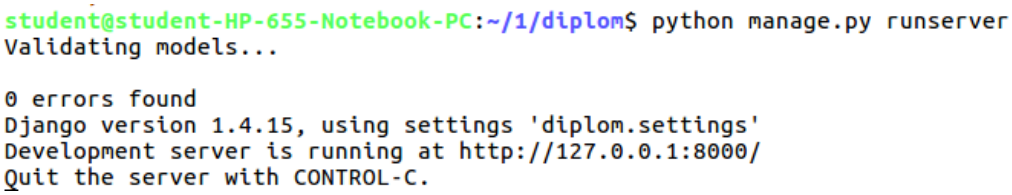

Рисунок 36 – Запуск сервера

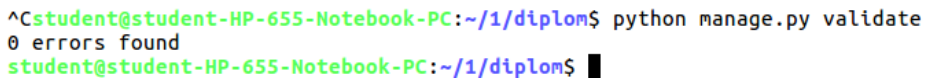

Рисунок 37 – Проверка на ошибки основного файла проекта *manage.py*

Исходный код файлов созданной базы данных представлен в Приложении Д.

3.5 Написание алгоритма поиска денотатов

Как описано выше, алгоритм поиска денотатов основан на модели данных. Каждая сущность модели – это таблица, каждое отношение – входные данные для этой таблицы. Для создания базы данных в пункте 3.4 был использован метод парсинга с использованием адаптированного под *Python* модуля *SQLite3*. Приступаем непосредственно к проектированию приложения. В уже созданную нами модель данных теперь необходимо «занести» ЭГ. Проанализировав все возможные способы, остановились на функциях *insert* и *select* модуля *SQLite3*. Очевидно, что функции модуля *SQLite3* и языка *SQL* имеют идентичный функционал, основное различие заключается в синтаксисе. В модуле он ориентирован на *Python*. Таким образом, с помощью *insert* мы занесем записи в таблицы, хранящие неизменяемые данные, такие как денотаты и слова, а с помощью *select* заполним таблицы связей между первичными данными.

Итак, создадим файл в редакторе *Atom* и сохраним под именем *data.py*.

В созданном файле пропишем функцию *example_data*, в которой будем заполнять поля таблицы с помощью объекта *cursor.execute*, который делает и получает запросы и функций *insert* и *select*. Так, как в создаваемой базе будет использоваться несколько подключений, необходимо воспользоваться методом соединения *.commit()*, которое не позволит БД отключаться после каждого соединения, до завершения их всех. Примеры создания записей и

запросов в файле *data.py* для основных таблиц изображены на рисунках 38-42.

```
def example_data(conn, cursor):
    cursor.execute('insert into lexema (infinitive, linguistic_characteristics_word)\
    values ("логический","");')
    conn.commit()
    cursor.execute('insert into lexema (infinitive, linguistic_characteristics_word)\
    values ("программирование","");')
    conn.commit()
    cursor.execute('insert into lexema (infinitive, linguistic_characteristics_word)\
    values ("организовываеть","");')
    conn.commit()
    cursor.execute('insert into lexema (infinitive, linguistic_characteristics_word)\
    values ("с","");')
    conn.commit()
    cursor.execute('insert into lexema (infinitive, linguistic_characteristics_word)\
    values ("помощь","");')
```

Рисунок 38 – Заполнение таблицы *lexema* с помощью функции *insert*

```
cursor.execute('insert into denotat (title)\
values ("логическое программирование");')
conn.commit()
cursor.execute('insert into denotat (title)\
values ("организовывано с помощью");')
conn.commit()
cursor.execute('insert into denotat (title)\
values ("математическая логика");')
conn.commit()
cursor.execute('insert into denotat (title)\
values ("реализовывано в");')
conn.commit()
cursor.execute('insert into denotat (title)\
values ("пролог");')
conn.commit()
cursor.execute('insert into denotat (title)\
values ("создает");')
```

Рисунок 39 – Заполнение таблицы *denotat* с помощью функции *insert*

```
conn.commit()
cursor.execute('insert into lexema_to_denotat (id_den, id_lexema, weight)\
values ((select id from denotat where title = "логическое программирование"),(select id from lexema where inf
conn.commit()
cursor.execute('insert into lexema_to_denotat (id_den, id_lexema, weight)\
values ((select id from denotat where title = "логическое программирование"),(select id from lexema where inf
conn.commit()
cursor.execute('insert into lexema_to_denotat (id_den, id_lexema, weight)\
values ((select id from denotat where title = "логическое программирование"),(select id from lexema where inf
conn.commit()
cursor.execute('insert into lexema_to_denotat (id_den, id_lexema, weight)\
values ((select id from denotat where title = "математическая логика"),(select id from lexema where infinitiv
conn.commit()
cursor.execute('insert into lexema_to_denotat (id_den, id_lexema, weight)\
values ((select id from denotat where title = "математическая логика"),(select id from lexema where infinitiv
```

Рисунок 40 – Заполнение таблицы *lexeme_to_denotat* с помощью функций *insert* и *select*

```
conn.commit()
cursor.execute('insert into dictionary (id_lexema, description)\
values ((select id from lexema where infinitive = "логический"),"");')
conn.commit()
cursor.execute('insert into dictionary (id_lexema, description)\
values ((select id from lexema where infinitive = "программирование"),"");'
conn.commit()
cursor.execute('insert into dictionary (id_lexema, description)\
values ((select id from lexema where infinitive = "организовывать"),"");')
conn.commit()
cursor.execute('insert into dictionary (id_lexema, description)\
values ((select id from lexema where infinitive = "с"),"");')
conn.commit()
cursor.execute('insert into dictionary (id_lexema, description)\
values ((select id from lexema where infinitive = "помощью"),"");')
```

Рисунок 41 – Заполнение таблицы *dictionary* с помощью функций *insert* и *select*

```
conn.commit()
cursor.execute('insert into relation (id_from, id_to, id_how, weight, direction)\
values ((select id from denotat where title = "логическое программирование"),\
        (select id from denotat where title = "математическая логика"),\
        (select id from denotat where title = "организовано с помощью"),1,1);')
conn.commit()
cursor.execute('insert into relation (id_from, id_to, id_how, weight, direction)\
values ((select id from denotat where title = "логическое программирование"),\
        (select id from denotat where title = "пролог"),\
        (select id from denotat where title = "реализовано в"),1,1);')
conn.commit()
cursor.execute('insert into relation (id_from, id_to, id_how, weight, direction)\
values ((select id from denotat where title = "пролог"),\
        (select id from denotat where title = "программа"),\
        (select id from denotat where title = "создает"),1,1);')
```

Рисунок 42 – Заполнение таблицы *relation* помощью функций *insert* и *select*

Помимо функции *example_data* в файле *data.py* содержится функция *example_tables*, которая хранит уже созданные нами таблицы модели данных и *example_text*, которая пригодится нам в процессе тестирования работы программы. Структура функции представлена на рисунке 43.

```
def example_tables(conn,cursor,table):
    if table == "rootobject":
        print("create table %s" % table)
        cursor.execute("create table rootobject (id integer primary key autoincrement not null
        return True
    if table == "relation":
        print("create table %s" % table) # Отношения
        cursor.execute("create table relation (id integer primary key autoincrement not null,
        return True
```

Рисунок 43 – Структура функции *example_tables*

Еще одна функция, которую необходимо прописать здесь же это функция для ввода тестируемого текста в БД. Назовем её *lexema_id_by_inf*.

Результат этой функции мы будем получать с помощью объекта *.fetchall()*. Описанная функция представлена на рисунке 44.

```
def lexema_id_by_inf(cur, lexema):
    cur.execute('select id from lexema where infinitive = "%s";' % (lexema))
    for row in cur.fetchall():
        return row [0]
    return 0
```

Рисунок 44 – Функция *lexema_id_by_inf*

Далее, прописываем функции *denotat_id_by_lexema_id* и *denotat_litle_by_ id*. С помощью этих функций программа выводит не *id* денотатов, а их имена в той форме, в которой они занесены в БД (рисунок 45).

```
def denotat_id_by_lexema_id(cur, id_lexema):
    ret = []
    cur.execute('select id_den from lexema_to_denotat where id_lexema = %s;'
    for row in cur.fetchall():
        ret += [row[0]]
    return ret

def denotat_title_by_id(cur, id):
    cur.execute('select title from denotat where id = %s;' % (id))
    for row in cur.fetchall():
        return row[0]
    return 0
```

Рисунок 45 – Функции *denotat_id_by_lexema_id* и *denotat_litle_by_ id*

Теперь, когда денотаты извлечены заносим в файл *data.py* функцию *relation_by_den_id*, которая найдет связи между найденными графами (в том случае если они есть). Вид функции представлен на рисунке 46.

```
def relation_by_den_id(cur, id_from, id_to):
    ret = []
    cur.execute('select id_how from relation where id_from = %s and id_to = %s;' % (id_from, id_to))
    for row in cur.fetchall():
        ret += [row[0]]
    return ret
```

Рисунок 46 – Функция *relation_by_den_id*

После заполнения всех данных, приступаем к созданию исполняемого файла, с помощью которого будем управлять созданной БД *denotator.db*. Назовем этот файл *z.py*. исполняемый файл содержит подключения к модулю *SQLite3*, функциям, содержащимся в файле *data.py* и инструментам *Python*, таким как *rutermextract* и *mystem* (рисунок 47).

```
import sqlite3
import sys
from data import example_data, \
                 example_text, \
                 example_tables, \
                 lexema_id_by_inf, \
                 denotat_id_by_lexema_id, \
                 denotat_title_by_id, \
                 relation_by_den_id
from rutermextract import TermExtractor
import pymystem3
```

Рисунок 47 – Подключения файла *z.py*

На рисунке 48 представлены функции вызовов инструментов *rutermextrac*t и *mystem.*

```
term_extractor = TermExtractor()
mystem = pymystem3.Mystem()
denotats = []
for term in term_extractor(example_text):
    lexemas = []
    for lexema in str(term.normalized).split(" "):
        lexema = mystem.analyze(lexema)[0]['analysis'][0]['lex']
        id_lexema = lexema_id_by_inf(cursor, lexema)
        lexemas += [id_lexema]
    for id_lexema in lexemas:
        id_denotat = denotat_id_by_lexema_id(cursor, id_lexema)
        denotats += id_denotat
```

Рисунок 48 – Функции вызова инструментов *rutermextrac*t и *mystem.*

Создаем список отношений «денотат-денотат». На рисунке 49 представлен цикл поиска вязи.

```
relations = []
for id_denotat_from in denotats:
    for id_denotat_to in denotats:
        rel_id = relation_by_den_id(cursor, id_denotat_from, id_denotat_to)
        try:
            assert rel_id[0] != None
            relations += [(id_denotat_from, rel_id[0], id_denotat_to)]
        except:
            pass
```

Рисунок 49 – Создание списка отношений

Для наглядности, выведем тестируемый текст перед списком «денотат-отношение-денотат». Для этого создадим функцию печати в файле *z.py* (рисунок 50).

```
print(example_text)
for relation in relations:
    print(
        denotat_title_by_id(cursor, relation[0]),
        denotat_title_by_id(cursor, relation[1]),
        denotat_title_by_id(cursor, relation[2])
```

Рисунок 50 – Функция вывода текста на экран

Полный листинг исполняемого файла и файла с данными представлен в Приложении Е.

3.6 Тестирование программы

Для проверки работы приложения возьмем случайный текст:

«Для программной реализации логическое программирование использует язык пролог. Программы на прологе включают в себя такие объекты, как факты, правила и вопросы. Факты в прологе это истинные значения, правила это заключения, а вопросы это цель. Правила состоят из предикатов».

Этот текст относится к предметной области «Информатика», по которой мы строили ЭГ, а именно к её разделу «Искусственный интеллект».

Вносим этот текст в функцию *example_text* в файле *data.py* (рисунок 51), сохраняем изменения.

```
example_text = u'Для программной реализации логическое программирование использует язык пролог.\
Программы на прологе включает в себя такие объекты, как факты, правила и вопросы. Факты в прологе это истинные'
значения, правила это заключения, а вопросы это цель. Обычно, правила состоят из предикатов.'
```

Рисунок 51 – Занесение тестового текста в БД

Сначала «пропустим» тестовый текст через инструмент rutermextract непосредственно в терминале. Это позволит нам сравнить адекватность работы алгоритма. На рисунке 52 выведен список извлеченных ключевых слов, среди которых встречаются такие как «такие объекты», «истинные значения», «язык», которые не являются денотатами в нашей модели. Если после выполнения нашего алгоритмы, созданная программа «найдет» их внутри себя, значит, алгоритм не корректен.

```
In [5]: from rutermextract import TermExtractor

In [6]: term_extractor = TermExtractor()
^[[A
In [7]: text = u'Для программной реализации логическое программирование использует
 язык пролог. Программы на прологе включает в себя такие объекты, как факты, прави
ла и вопросы. Факты в прологе это истинные значения, правила это заключения, а воп
росы это цель. Обычно, правила состоят из предикатов.'

In [8]: for term in term_extractor(text):
    print(term.normalized, term.count)
   ...:
пролог 3
правило 3
факты 2
вопросы 2
такие объекты 1
программная реализация 1
логическое программирование 1
истинные значение 1
язык 1
цель 1
программа 1
предикаты 1
заключение 1
```

Рисунок 52 – Извлеченные ключевые слова

Открываем терминал *Ubuntu* и запускаем исполняемый файл *z.py*. Перед запуском лучше пересоздавать БД, с целью избежать внутренних ошибок программы. На экране появится процесс обновления таблиц, а затем терминал выведет на экран тестируемый текст и список «денотат-отношение-денотат» (рисунок 53).

```
student@student-HP-655-Notebook-PC:~/Загрузки$ python3 z.py
create table rootobject
create table relation
create table denotat
create table lexema
create table lexema_to_denotat
create table argument
create table scopos
create table question
create table text_problematics
create table dictionary
create table collocation
create table text
create table subject_area
create table key_words
create table figure_of_speech
create table sentence
create table research_topic
example_data_loaded
/usr/lib/python3.5/re.py:203: FutureWarning: split() requires a non-empty patter
n match.
  return _compile(pattern, flags).split(string, maxsplit)
Для программной реализации логическое программирование использует язык пролог. П
рограммы на прологе включает в себя такие объекты, как факты, правила и вопросы.
 Факты в прологе это истинные значения, правила это заключения, а вопросы это це
ль. Обычно, правила состоят из предикатов.
пролог создает программа
правило является заключение
вопрос является цель
программа состоит из правило
программа состоит из факт
программа состоит из вопрос
```

Рисунок 53 – Результат запуска исполняемого файла *z.py*.

ЗАКЛЮЧЕНИЕ

Предположение о том, что в качестве модели понимания основной мысли текста может быть использована денотатная модель, представленная в графовой форме, подтвердилась.

В ходе работы были исследованы принципы, методы и средства денотативного анализа текстов, а так же способы создания графового представления структуры текста. Была построена модель данных для автоматизированной системы, основанная на ранее изученном теоретическом материале. На основе модели данных, написан ЭГ для предметной области «Информатика», а затем, с помощью построенного граф, протестированы тексты, принадлежащие выбранной предметной области. На основе полученных результатов, построены денотатные графы для каждого из текстов. С помощью этого подтвердили адекватность спроектированного ЭГ. Убедившись в правильности построенной инфологической модели и на её основе, разработали программный код для хранения денотатной структуры в памяти компьютера. Создали БД, которую заполнили сущностями из разработанного ЭГ. Затем, создали алгоритм для вывода структуры «денотат-отношение-денотат». Проведено тестирование текста для выбранной предметной области «Информатика» для проверки работы программного кода и подтверждена адекватность приложения. Все поставленные задачи выполнены.

В дальнейшем, разработанный алгоритм будет использован для создания автоматизированной системы реферирования, которая будет основана на методе денотативного анализа текста.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Новиков А.И., Нестерова Н.М. Реферативный перевод. М.: Академия наук СССР Институт языкознания, 1991.

2. Нестерова Н.М., Герте Н.А. Реферирование как способ извлечения и представления основного содержания текста // Вестник пермского университета, российская и зарубежная философия. 2013. №2(24).

3. Герте Н.А. Денотативная модель реферативного специализированного перевода: автореф. дис. ... канд. филол. наук: 10.02.19. Пермь, 2016.

4. Ягунова Е.В., Пивоварова Л.М. Опыт автоматического извлечения и классификации на материале новостных текстов // Природа колокаций в русском языке.. М.: Сб. НТИ, 2010.

5. Н.А. Калугин, А.Н. Калугин Элементы теории графов. Самара: Самар.гос.аэрокосм.ун-та, 2013

6. Способы хранения графов в памяти компьютера // Dexperix URL: http://dexperix.net/articles/alg/xpanenie_grafa%2Bnpi%2Bru.html (дата обращения: 1.06.2017).

7. Обзор DVYGUN Smart Search - персональной поисковой системы // PegАРД URL: http://www.ixbt.com/soft/dvygun-smart-search.shtml (дата обращения: 28.05.2017).

8. Некоторые методы автоматического анализа естественного языка, используемые в промышленных продуктах // MyTTS URL: http://mytts.forum2x2.ru/t182-topic (дата обращения: 10.06.2017).

9. TextAnalyst - работа с текстовой информацией // Intarface.ru URL: http://www.interface.ru/home.asp?artId=33240 (дата обращения: 10.06.2017).

10. Ступин В.С. Система автоматического реферирования методом симметричного реферирования // Доклады международной конференции Диалог 2004. М: 2004.

11. ПС для обработки текста // tpl-it URL: http://tpl-it.wikispaces.com (дата обращения: 10.06.2017).

12. Atom — текстовый редактор // Soft-file URL: https://soft-file.ru/atom-tekstovyj-redaktor/ (дата обращения: 13.06.2017).

13. Graphviz — пакет утилит для автоматической визуализации графов Источник: http://pro-spo.ru/-linux-e/1925-graphviz- // pro-spo.ru URL: http://pro-spo.ru/-linux-e/1925-graphviz- (дата обращения: 13.06.2017).

14. Ubuntu // Ubuntu URL: http://ubuntu.ru/ (дата обращения: 13.06.2017).

15. Python // Python URL: https://www.python.org/ (дата обращения: 13.06.2017).

16. Документация Django // djBook.ru URL: https://djbook.ru/rel1.7/ (дата обращения: 13.06.2017).

17. Beautiful Soup Documentation // Beautiful Soup Documentation URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата обращения: 13.06.2017).

18. Rutermextract // Python URL: https://pypi.python.org/pypi/rutermextract (дата обращения: 13.06.2017).