Московский государственный университет им. М.В. Ломоносова

Черноморский филиал

Факультет Компьютерной математики

Кафедра программирования

**Разработка и исследование методов автоматической коррекции орфографических ошибок при машинном чтении текстов.**

Дипломная работа

Выполнил студент

группы ПМ-501

П.С. Сухотюк.

Руководитель

Е.А. Бутаков.

Севастополь

2008

**Аннотация.**

В данной работе рассматривается проблема коррекции грамматических ошибок в сканированных текстах естественного языка. В связи с этим в работе поставлены следующие задачи:

1. Исследовать существующие методики диагностики и коррекции одиночных грамматических ошибок в текстах русского языка.
2. Провести сравнительный анализ методов.
3. Улучшить метод диагностики и коррекции одиночных грамматических ошибок в текстах русского языка, основанный на морфологическом анализе, за счет уменьшения количества вариантов коррекции.

Реализован программный комплекс диагностики и коррекции орфографических ошибок в сканированных текстах русского языка. Уменьшено количество вариантов коррекции в методе, основанном на морфологическом анализе, за счет применения частотного словаря, что позволило повысить процент корректно исправленных слов в 4 раза.

**Введение.**

Проникновение электронной информации во все слои человеческой деятельности заставляет библиотечные учреждения активно осваивать современные информационные технологии, внедрять автоматизированные системы, создавать электронные библиотеки, развивать Интернет-услуги. Цивилизованный мир однозначно воспринимает эти изменения как одну из характерных тенденций общества будущего, как систему, облегчающую доступ пользователей к информационным ресурсам.

Нарастание потоков электронной информации во всех сферах человеческой деятельности резко повышает статус информационных ресурсов. Чтобы полно и оперативно удовлетворять информационные запросы современного пользователя, владельцы ресурсов должны быть в центре внедрения современных технологий и использования электронной информации. Реальностью сегодняшнего дня стали электронные издания, число которых постоянно увеличивается. Библиотеки, не имеющие в фонде тех или иных электронных изданий и предоставляющие к ним доступ через Интернет, уже с полным на это основанием включают их библиографические описания в свои каталоги и предоставляют их пользователям.

Главная особенность – самостоятельность пользователя в работе с информационными ресурсами. Это требует обеспечения определенного уровня обслуживания и заставляет владельцев таких ресурсов более тщательно относится к системе автоматизации, к возможностям информационных технологий, внедряемых в практику. [1]

При создании различных по назначению баз данных производится ввод текстовой информации, осуществляемый двумя способами – набором вручную или сканированием. В обоих случаях возможны орфографические ошибки. Положение орфографических ошибок в классификации ошибок русского языка дано в [2].

Современные текстовые редакторы (например, MS Word), как правило, проводят автоматизированную диагностику орфографических ошибок. Это требует вмешательства пользователя при коррекции. Автоматическая коррекция орфографических ошибок может быть более эффективным средством минимизации опечаток и их исправлений при создании текстовых файлов.

**Цель и постановка задачи.**

**Определение:** Буква – графический знак, который сам по себе или в сочетании с другими знаками используется для обозначения на письме звуков, фонем, их основных вариантов и их типичных последовательностей.

**Определение:** Словоформа – слово в заданной грамматической форме.

**Определение:** Лексема – (от греческого lexis – слово, выражение), единица лексического уровня языка, слово во всей совокупности его лексических значений.

**Определение:** Текст — это упорядоченный набор [предложений](http://ru.wikipedia.org/wiki/Предложение), предназначенный для того, чтобы выразить некий [смысл](http://ru.wikipedia.org/wiki/Смысл).

**Определение:** Язы́к — система звуков, знаков, предназначенная для фиксации, переработки и передачи сведений от одного субъекта к другому.

Орфографически ошибочным словом называется буквенная цепочка, полученная некоторым преобразованием из словоформы некоторой лексемы, принадлежащей естественному языку. Под исправлением ошибки в таком слове понимается установление исходной словоформы. Исходная словоформа определяется неоднозначно. В данной постановке задача исправления ошибки называется также задачей полного словарного исправления. Результатом попытки исправления ошибки в пределах некоторого класса преобразований может быть также установление невозможности ее исправления, то есть отсутствия в словаре словоформы, из которой данная цепочка получается каким-либо преобразованием заданного класса [3].

Опечатками называются ошибки, связанные с поверхностным, буквенным представлением слова, то есть с искажениями непосредственно буквенной цепочки, представляющей словоформу [3].

С целью объяснения и исправления ошибок выделяется некоторый класс элементарных искажений. Например, замена одной буквы на другую, перестановка гласной буквы через согласную, сдвиг руки на одну позицию при набивке части слова на клавиатуре. Сложными называются ошибки, являющиеся комбинацией нескольких элементарных. Например, замена двух букв в одном слове. Элементарное искажение называется локальным, если оно по определению затрагивает небольшой отрезок буквенной цепочки, например, не больше трех букв. Цепочка называется словом с одиночной ошибкой, если она содержит только одно элементарное искажение [3].

Одним из важнейших классов элементарных искажений является класс одиночных ошибок, включающий в себя:

1. Замена одной буквы на другую (83%).
2. Удаление буквы (8%).
3. Добавление лишней буквы (4,5%).
4. Перестановка двух букв (4,5%).

Одиночные ошибки данного класса составляют около 90% всех искажений в текстах. К таким опечаткам может быть сведено и большинство орфографических ошибок. В последнем случае ошибка может быть правильно исправлена и проинтерпретирована, но ее объяснение не является правильным. Например, *сильнейшвй* и *сильнейшый*. [3]

Целью дипломной работы является разработка метода коррекции орфографических ошибок в сканированных текстах естественного языка. В связи с этим в работе поставлены следующие задачи:

1. Исследовать существующие методики диагностики и коррекции одиночных орфографических ошибок в текстах русского языка.
2. Провести сравнительный анализ методов.
3. Улучшить метод диагностики и коррекции одиночных орфографических ошибок в текстах русского языка, основанный на морфологическом анализе, за счет уменьшения количества вариантов коррекции.

**Обзор существующих решений рассматриваемой задачи.**

Коррекцию ошибок можно условно разделить на два этапа: диагностика и непосредственное исправление ошибки. Таким образом, процесс коррекции ошибок в буквенной цепочке выглядит следующим образом:

1. Составление словаря, состоящего из токенов (в [лингвистике](http://ru.wikipedia.org/wiki/Лексема_(лингвистика)) — слово как самостоятельная единица языка, рассматриваемая во всей совокупности своих форм и значений), имеющих значение для алгоритма диагностики (не обязательно точное разбиение на морфемы).
2. Выделение во входной цепочке символов токена, предположительно содержащего ошибку.
3. Коррекция ошибки в токене с помощью алгоритма исправления ошибки.

На практике полностью отделить диагностику от коррекции ошибок возможно не всегда.

**Метод n-грамм.**

N-грамма – это цепочка символов (букв), которые могут встречаться в словах, принадлежащих определенному языку. Метод основан на том факте, что в каждом флективном языке существует строго ограниченный набор допустимых сочетаний символов.

Алгоритм был весьма популярен виду своей простоты и относительно неплохого процента успешного определения ошибочных слов. До сих пор широко используется реализация данного метода в продукте Punto Switcher [16]. Его задача состоит в том, чтобы определять на какой раскладке клавиатуры (русская или английская) набрано данное слово.

Однако данный метод сильно зависит от длины n-граммы. Чем длиннее n-грамма, тем точнее будут результаты. Однако здесь имеется обратная сторона: увеличение длины n-граммы увеличивает объем словаря и, как следствие, время работы алгоритма. Например, в работе [13] приводится цифра в 7000 3-грамм, в программе Punto Switcher [16], по словам разработчиков, – 25219 2-, 3-, 4-грамм. Однако проблема очевидна – увеличение количества букв в n-грамме резко увеличивает словарь.

Для наиболее корректной диагностики орфографических ошибок необходимо более корректно учитывать структуру и особенности языка. Таковым является морфологический анализ.

**Морфологический анализ.**

Основным недостатком алгоритмов использующих словарь является размер словаря. Например, [4] содержит 52 000 слов, [5] – 100 000 слов, а словарь [10] – 173 780 слов (3 028 433 словоформы). Из-за больших объемов данных перебор всех слов крайне не желателен.

Согласно исследованиям [5] примерно 5000 морфем позволяют построить около 50000 слов. Для украинского языка соотношение примерно такое же [6]. Учет такой информации позволяет разработать словарь с 10 кратным сжатием.

**Морфологический анализ** — процесс выявления морфологических составляющих слова. **Морфема** — наименьшая [языковая единица](http://ru.wikipedia.org/w/index.php?title=Языковая_единица&action=edit), обладающая [значением](http://ru.wikipedia.org/wiki/Значение) (по определению, данному американским лингвистом [Леонардом Блумфилдом](http://ru.wikipedia.org/w/index.php?title=Леонард_Блумфилд&action=edit) в [1933](http://ru.wikipedia.org/wiki/1933)) [7].

Классификация морфем:

1. **Корень** - основная значимая часть слова, непроизводная основа слова. Корень является обязательной частью любого слова — не существует слов без корня. Корневые морфемы могут образовывать слово как в сопровождении аффиксов, так и самостоятельно [8].
2. **Аффикс** — вспомогательная часть слова, присоединяемая к корню и служащая для словообразования и выражения грамматических значений. Аффиксы не могут самостоятельно образовывать слово — только в сочетании с корнями. Аффиксы, в отличие от некоторых корней (как, например, *какаду*), не бывают единичными, встречающимися только в каком-то одном слове [8].

* Префикс – аффикс, предшествующий корню (приставка).
* Постфикс – аффикс, следующий за корнем (суффикс, окончание).
* Интерфикс – аффикс, соединяющий несколько корней.

Пример: слово *выстиранный* имеет корень *–тир–*, приставки *вы–*, *с–*, суффиксы *–а–*, *–нн–* и окончание *–ый*.

В сложных словах присутствует несколько корней, соединенных соединительной гласной: *пар–о–ход*, *пыл–е–сос*.

Учет морфемной структуры слов позволяет компактно представлять совокупность словоформ, группируя их в словообразовательные гнезда. Такое представление реализуется в виде словаря морфем, содержащего три части – корневую, префиксальную и суффиксальную [5].

В корневой части словаря корни расположены в лексикографическом порядке. В словарной статье под каждым заглавным корнем приводятся однокоренные слова, расчлененные на морфемы. Они расположены следующим образом. Во главе гнезда ставится корневое слово, причем на первом месте обычно располагается существительные с нулевым или выраженным окончанием, на втором – неизменяемые части речи типа наречий, междометий. За корневыми словами идут беспрефиксные слова, имеющие суффиксы (лексикографически упорядоченные), затем следуют по алфавиту префиксальные слова первого префикса в слове.

Если при одной и той же основе может быть несколько производных, то непосредственно после основы слова, разделенной на морфемы, указываются под различными верхними индексами окончания, расположенные в алфавитном порядке, которые, присоединяясь к основе, образуют различные слова, например, под корнем –*зл–* даются слова *√–о1, –ой2*, т.е. слова *зло*, *злой*.

В префиксальной части словаря под каждым заглавным префиксом дается в алфавитном порядке перечень всех аффиксальных окружений корня, в которых встречается данный префикс; рядом перечисляются все корни, в которых встречается данный префикс; рядом перечисляются все корни, употребляющиеся в соответствующем окружении. Слова, начинающиеся прямо с корня приводятся в начале.

Использование корневой части [5] как основы электронных словарей на данный момент является стандартом в системах морфологического анализа [9].

Рассмотрим алгоритм коррекции ошибок на основе морфологического анализа.

Этап 1. На первом этапе предполагается, что в рассматриваемом слове нет префикса и поэтому в нем пытаемся выделить одну из основ с нулевым префиксом. Если это не удается, то либо в слове есть ошибка, либо в слове есть префикс (этап 2). Сначала проверяется гипотеза об ошибке в основе. Для этого:

1. производятся все возможные разбиения анализируемого слова на основу и окончание;
2. для каждой из полученных основ находится обратное пол­ное преобразование, т.е. генерируются все основы, могущие по­родить данную в результате одиночной ошибки.

Если сгенерированная основа есть в словаре, то она проверяется на совмести­мость (принадлежность к флективному классу) с окончанием ана­лизируемого слова. Если сгенерированная основа не совместима с окончанием анализируемой основы, то она из рассмотрения исключается и генерируется следующая основа; если же сгенерированная основа совместима с окончанием, т. е. она находится в одном флективном классе с анализируемой основой, то она заносится в список кандидатов и совершается переход к генерации следующей основы. По окончании процесса генерации проверяется число основ в списке кандидатов. Если в этом списке окажется больше одной основы, то после присоединения окончаний (разумеется, одинаковых, совпадающих с окончанием анализируемого слова), все они выдаются на экран. Если в списке кандидатов - только одна основа, то после присоединения к ней окончания замещаем этой словоформой анализируемую. Ошибка исправлена. Если же в списке кандидатов не оказалось ни одной осно­вы, то это значит, что окончание было выбрано неверно, нужно испытать другое окончание. Если в результате перебора всех до­пустимых пар основа-окончание не удалось найти ни одного кан­дидата, то это значит, что в слове есть префикс, переходим к этапу 2.

Вернемся теперь к началу этапа и рассмотрим случай, когда в слове удалось выделить основу с нулевым префиксом. В этом случае нужно проверить соответствие окончания анализируемого слова флективному классу основы. Проверка на совместимость основы с окончанием может дать следующие результаты: либо подтвердится, что основа определена правильно, либо обнару­жится ошибка в окончании и, возможно, будет исправлена. Сложность исправления ошибки в окончании состоит в том, что в ре­зультате ошибки в окончании возможна замена одного окончания на другое из той же самой группы окончаний, определяющие флек­тивный класс.

Этап 2. На втором этапе проверяется наличие в словоформе префикса. Сначала предполагается, что в префиксе нет ошибки. Начиная с префикса наибольшей длины с той же начальной буквой, что и слово, пытаемся выделить префикс в словоформе. Если это удалось, то среди основ, совместимых с данным префиксом, пыта­емся найти ту, которая входит в анализируемую словоформу. Если это удается, то осуществляется проверка окончания на принад­лежность к группе окончаний, определяющих флективный класс найденной основы. Положительный результат проверки свиде­тельствует о том, что в слове нет ошибки, поэтому осуществля­ется переход к анализу следующего слова; отрицательный резуль­тат проверки свидетельствует об ошибке в окончании. Если перебор основ, совместимый: с данным префиксом не дал результата, то это значит, что либо в основе есть ошибка, либо префикс выбран ошибочно. Сначала обращаемся к модулю проверки гипотезы об ошибке в основе (точка (а) на блок-схеме). Если не удалось найти ошибку в основе, то возврат из модуля происходит в точку (в) - выбор следующего префикса. Если перебор всех префиксов с той же начальной буквой, что и слово не дает результата, то осуществляется переход к этапу проверки гипотезы об ошибке в префиксе (метка (ж) на блок-схеме).

Этап 3. Проверку гипотезы об ошибке в префиксе начинаем с того, что пытаемся отделить префикс от основы. Для этого последовательно убираем начальные буквы слова, и для каждого начального отрезка из 1, 2, ..., k букв, находим обратное пол­ное отображение (ОПО). Каждый элемент этого отображения рассматривается как возможный правильный префикс. Эта последо­вательность символов сравнивается со всеми префиксами языка. В случае совпадения начинается перебор основ, совместимых с этим префиксом, и осуществляется проверка вхождения основ в анали­зируемое слово. Если найдется основа, входящая в анализируемое слово, то проверяется принадлежность окончания флективному классу этой основы. При положительном результате проверки най­денный префикс присоединяется вместо удаленных букв, анализи­руемое слово заменяется полученным, осуществляется переход к анализу следующего слова. Если окончание не принадлежит флек­тивному классу, то считается, что в окончании ошибка, а это значит, что в слове более одной ошибки.

Вернемся теперь к случаю, когда элемент ОПО не совпал ни с одним из префиксов языка. В этом случае осуществляется пере­ход к генерации следующего элемента ОПО. Если генерация всех элементов ОПО для начального отрезка словоформы длиной s не дала элемента, совпадающего с одним из префиксов, то длина от­резка, рассматриваемого как возможный префикс (но с ошибкой!), увеличивается на единицу и процесс повторяется.

Эвристически предполагается, что в любом слове помимо префикса есть еще не менее трех букв. Поэтому, если в остатке словоформы после удаления (s) букв окажется менее трех букв, то либо слова нет в словаре, либо в нем более одной ошибки [3].

Тестирование данного алгоритма в [3] проводилось следующим образом:

1. Вводится текст в ЭВМ.
2. Последовательно выбираются отдельные слова этого текста.
3. С помощью генератора случайных чисел генерируется случайное число r из интервала [1, …, p], где p – число букв в слове. Это слово принимается за номер позиции в слове, в которую будет внесена ошибка. Поскольку рассматривается 4 типа ошибок и считается, что ошибки различных классов равновероятны, то снова генерируется случайное число из интервала [1, …, 4], которое определяет тип вносимой орфографической ошибки. Наконец, третье случайное число генерируется в случае, если случится ошибка типа “замена буквы”. Оно определяет, какая буква будет поставлена на место буквы, стоящей на месте r.
4. Вносится ошибка соответствующего типа. Выбирается из следующее слово. По окончании перебора всех слов процесс генерации текста с ошибками заканчивается.
5. Запускается механизм обнаружения и коррекции ошибок. Оценивается доля ошибок, исправляемых автоматически и доля ошибок, для исправления которых потребовалось вмешательство оператора – это ошибки в некоторых окончаниях.[3]

Использование одного морфологического анализа для изолированного слова, как правило, недостаточно для автоматической проверки текстов русского языка, так как он не дает точного результата на больших словарях. Для этих целей необходимо использование комплексного анализа рядом стоящих слов (словосочетаний) или целых предложений – методов корпусной лингвистики.

**Корпусная лингвистика.**

Корпусной лингвистикой называется направление изучения языка, которое включает в себя все процессы, связанные с обработкой, использованием и анализом корпусов текстов, доступных для «машинного» чтения. Термин «корпусная лингвистика» является относительно новым, он обозначает методологию, основанную на примерах использования языка в реальной жизни. В настоящее время эффективность корпусной лингвистики тесно связана с развитием вычислительной техники. Термин «корпус» обычно обозначает либо текст, доступный для машинной обработки, либо конечный набор текстов, максимально отображающий многообразие языковых явлений. Обработка корпусов текстов ведется с помощью специальных программ – анализаторов, определяющих синтаксическую структуру текстов. Анализ обычно начинается с выделения в тексте основных морфосинтаксических категорий, затем исследуется взаимосвязь слов на более высоком синтаксическом уровне. Такой анализ применяется, например, для аннотирования текстов. [12]

Анализаторы условно можно разделить на два типа: осуществляющие полный и осуществляющие частичный разбор синтаксической структуры предложений. В первом случае ставится задача получить настолько детализированную структуру, насколько это вообще возможно. Во втором случае используются более грубые синтаксические категории, и, например, могут игнорироваться внутренние структуры фрагментов определенных типов, несущественных для решаемой задачи. [12]

Одним из важных объектов изучения в корпусной лингвистике является словосочетание (collocation). Этот термин обозначает комбинацию слов, с определенной вероятностью встречающихся рядом; если словосочетание появляется в тексте достаточно часто, его называют значимым словосочетанием. Слова в словосочетании могут быть разделены другими словами, но число этих слов обычно не превышает четырех. Выделение словосочетаний в тексте имеет существенное практическое значение для составления словарей, обработки текстов на естественном языке и обучения языкам. [12]

Одной из типичных проблем корпусной лингвистики является полнота корпуса. Часто обработать все тексты из определенной области не представляется возможным, и тексты в корпус приходится включать выборочно, ориентируясь на достижение оптимального соотношения между репрезентативностью и объемом текстов. Тем не менее установлено, что характерной особенностью выборочных исследований достаточно обширной области является их заведомая неполнота. Разнообразие элементов может быть столь велико, а распределение их по численностям столь неравномерно, что ни в одну реально мыслимую выборку все они попасть решительно не в состоянии. [12]

Достаточно сказать, что объем подавляющего большинства лексических выборок (в количестве словоупотреблений) значительно меньше оценок объема словаря языка. Эта принципиальная недостаточность выборочных наблюдений приводит к тому, что зависимость вида ранг - частота при изменении объема наблюдений меняет свою форму. Обычно исследователи игнорируют этот момент, просто считая зону малых частот и высоких рангов статистически недостоверной. При этом может теряться возможность прогноза формы ранговой кривой на другие объемы и сравнения выборок разных объемов [12].

Одним из традиционных применений методологии корпусной лингвистики является построение статистических словарей, например, словарей отдельных поэтов и писателей (можно отметить словарь языка Шекспира[14]; в России одним из первых опытов подобного рода был словарь языка Пушкина [15]). В 1960-1970-х г.г. наблюдалось широко распространенное увлечение частотными словарями, особенно в связи с педагогическими и информационными приложениями. От очень скромных по объему (400 тыс. словоупотреблений) лексикография шагнула к рубежу в 1 млн. словоупотреблений, а затем и к новым рекордам — максимально дифференцированный словарь американских текстов для школы содержит более 5 млн. словоупотреблений, а словарь, созданный Институтом французского языка, опирается на корпус литературных текстов объемом более 70 млн. словоупотреблений. Затем наступает кризис: электронные корпусы текстов продолжают множиться и увеличиваться по объему (в некоторых из них счет идет уже на сотни миллионов словоупотреблений), но не видно новых частотных словарей, которые были бы созданы на основе этих корпусов. Это отчасти объясняется тем, что процесс лемматизации (выделения основ слов и «начальных» форм) не поддается алгоритмам на сто процентов. Доля ручного вмешательства хотя и уменьшается относительно, но продолжает расти абсолютно. При росте объема текстового корпуса в 100 раз, объем ручного труда при постредактировании возрастет примерно в 10 раз. [12]

Статистические словари и таблицы могут содержать не только отдельные слова, но и словосочетания, предоставляя богатый материал для изучения комбинаторики слов в тексте. В приводятся примеры использования различных словосочетаний в текстах Достоевского, в частности, предлога *до*. Анализ правых соседей в данном случае обнаруживает свидетельства употребления этого предлога в пространственном значении (*до ворот*, *до дверей*, *до дивана*, *до дому*, *до кареты*, *до квартиры*, *до места*, *до Москвы*, *до передней*, *до печки*, *до поворота*, *до порога*, *до угла*). Несравненно чаще обнаруживаются темпоральные употребления (*до вечера*, *до времени*, *до завтра*, *до ночи*, *до рассвета*, *до свету*, *до сегодня*, *до утра*). Иногда маркером времени выступают какие-то привычные символы (до гроба, до могилы), либо обозначения конкретных событий (*до обеда*, *до отъезда*, *до приезда*, *до прихода*, *до свадьбы*). Очень часто предлог ассоциируется с идеей степени и предела, что уже многое говорит об особенностях семантики Достоевского (*до глубины*, *до конца*, *до крайности*, *до нитки*, *до основания*, *до последнего*, *до самого*, *до того*, *до чего*; а с другой стороны – *до малейшей*, *до мелочи*, *до одного*, *до одной*). [12]

**Синтаксические анализаторы.**

В самом начале работ по анализу естественного языка с помощью вычислительных машин не существовало ни синтаксических теорий, достаточно четких и пригодных для компьютерных реализаций, ни тем более формализованных семантических описаний естественно-языковых структур. Программы разбора представляли собой коллекции "упакованных программ", "помеченных подпрограмм", и т.п., которые постоянно разрастались по мере того, как грамматики расширялись для обработки все более сложных предложений, и вследствие такого усложнения затруднялось их взаимодействие. Далее программы обработки естественного языка развивались по двум основным направлениям. В первом из них использовался процесс сопоставления образцов (pattern matching) для выявления информации, а традиционный синтаксис при этом игнорировался. Второй подход был связан с рассмотрением некоторого ограниченного подмножества естественного языка на основе одного из вариантов контекстно-свободной грамматики. Обзор ранних программ грамматического разбора представлен в [12] .

Возникали и развивались многие лингвистические теории, рассматривавшие синтаксис как особый аспект, не связанный с семантикой. Язык трактовался как способ организации цепочек абстрактных символов, а его структура как набор правил манипулирования символами. Основным методом американского структурализма являлся и продолжает оставаться анализ непосредственно составляющих (в Европе в большей степени уделялось внимание грамматикам зависимостей), суть которого состоит в установлении дистрибутивного подобия отдельных языковых единиц, и составляющие рассматриваются как классы эквивалентности. Первой формализацией идеи иерархии составляющих были работы Хомского. С этого момента большая часть генеративистских лингвистических теорий основывалась (пусть даже частично) на контекстно-свободных грамматиках [13, 14], теория управления и связывания [15, 16, 17] и другие работы, причем многие из них используют схематические контекстно-свободные шаблоны, известные как Х-штрих схемы. Вскоре после Хомского контекстно-свободная грамматика была заново открыта Бэкусом, и независимо от него Науром и применена ими для описания языка АЛГОЛ. Решение рассматривать синтаксис на основе трансформационной грамматики Хомского было попыткой некоторых исследователей воспроизвести глубинную структуру предложения, которая затем могла бы быть проанализирована контекстно-свободным базовым компонентом. После ранних работ было создано множество вычислительных моделей обработки естественного языка на основе контекстно-свободных грамматик, поскольку они обеспечивали построение эффективных алгоритмов разбора в соответствии с этими грамматиками.[12]

Другим классом грамматических теорий, основывающихся не на контекстно-свободных грамматиках, а на взаимоотношениях между словами, являются грамматики зависимостей; грамматика слов, грамматика ограничений. Грамматики зависимостей очень широко применяются в современных статистических системах грамматического разбора, поскольку решающая роль непосредственных отношений между словами становится все более очевидной. В тесной связи с грамматиками зависимостей находится идея разработки обучаемых систем, содержащих глагольные фреймы, учитывающие все возможные аргументы глаголов и их характеристики, включая вероятностные оценки их появления. Одним из таких проектов является система баз данных WordNet. [12]

Подобные идеи лежат в основе системы ЯРАП, предназначенной для автоматического (машинного) перевода. Процесс перевода состоит из трех стадий - анализ — межъязыковые операции — синтез; нас в данном случае интересует прежде всего процесс внутриязыкового анализа. Подход выражается принципом словоцентризма в той его трактовке, согласно которой все структурные и другие характеристики предложений и словосочетаний могут быть заданы как свойства словоформ. Лингвистические знания, являющиеся результатом внутриязыкового анализа переводимого текста (в том числе, например, сведения о лексико-синтаксических отношениях в структуре входящих в него словосочетаний), должны быть заданы в конечном итоге в описании отдельных словоформ и распределены в них по составляющим эти словоформы лексико-морфологическим элементам.

Принципиальным теоретическим недостатком такого подхода является, видимо, невозможность адекватного описания языковых конструкций, в которых отсутствует явно выраженная главная словоформа. В этом случае приходится либо искусственно выбирать главное слово в словосочетаниях, либо считать все словосочетание единой лексической единицей. Главной же проблемой применения этого подхода на практике является необходимость формирования огромного количества словарных статей, отражающих все возможные варианты синтаксических отношений для каждого слова.

**Лингвистические процессоры интегрального и модульного типа.**

Одним из центральных вопросов является вопрос о том, каковы задачи и место синтаксического этапа анализа в когнитивной модели в целом, и прежде всего о взаимоотношении синтаксического и более высоких уровней (семантического и прагматического) и вообще о целесообразности разделения этих уровней в модели понимания естественного языка. Существуют два принципиально различных подхода. Один из них предполагает такое устройство системы, при котором каждому уровню лингвистического анализа соответствует отдельный компонент (модуль) системы: морфологический, синтаксический, семантический. Системы модульного типа допускают разные схемы взаимодействия модулей (последовательная работа, параллельный перемежающийся анализ), - это не меняет существа дела: синтаксис и семантика обрабатываются в системе разными механизмами. При этом синтаксический уровень анализа входного текста представлен в системе в явном виде: он выделен в отдельный блок, преобразующий текст в его синтаксическое представление. [12]

Другая стратегия анализа реализована в системах интегрального анализа. Синтаксический и семантический анализаторы (а зачастую и анализатор прагматического уровня) слиты в единый блок. Система ориентируется сразу на формирование на основе текста достаточно богатых концептуальных структур, а не на постепенное углубление понимания, как это имеет место в системах модульного типа. В системе интегрального типа не предусматривается формирование синтаксического представления входного текста. Синтаксическая информация используется фрагментарно и лишь как вспомогательная. Принцип интегрального анализа успешно применяется в основном в системах искусственного интеллекта, работающих в узкой предметной области. [12]

Другим важным вопросом является вопрос о том, насколько развитым и "семантикализированным" должен быть синтаксический этап анализа. Это проявляется в разной степени дифференцированности синтаксических отношений, в разной глубине интерпретации синтаксических отношений, а также в широте привлечения семантической информации при построении синтаксической структуры входного предложения. [12]

Можно признать, что анализаторы модульного типа с отдельным синтаксическим компонентом, не использующим семантической информации, располагаются в своей экологической нише; их применение оправдано там, где семантический компонент недостаточно развит сам по себе, и значительная часть общей нагрузки приходится на синтаксис. Такими системами является большинство полных систем машинного перевода. [12]

В то же время системы интегрального типа, как правило, лишь в незначительной степени используют синтаксический компонент. Существует несколько причин такого способа использования синтаксической информации; главными из них, по-видимому, являются следующие:

* + Отбрасывание синтаксически неправильных конструкций на этапе синтаксического анализа, что существенно ограничивает область применения всей системы. При этом возникает следующая дилемма: чем точнее описывать синтаксис языка, тем качественнее будет результат синтаксического анализа, но более строгие ограничения будут накладываться на входной язык, на степень его грамматичности.
  + Трудности, возникающие при стремлении объединить или связать описания синтаксиса и семантики языка, что приводит к тому, что один компонент, семантический, получает привилегированное положение, а синтаксическая информация в очень ограниченном варианте привязывается к описанию семантики.

На современном этапе развития компьютерной лингвистики можно констатировать, что системы модульного типа в чистом виде уступают свое место интегральным системам, и в связи с этим все более остро встают проблемы учета синтаксически неправильных конструкций в описании синтаксиса языка и обеспечения возможности описания синтаксической и семантической информации по единым принципам.[12]

**Сетевые грамматики.**

Синтаксический компонент в системах модульного типа может быть реализован различными способами. Выбранный способ должен удовлетворять двум условиям:

* + простота и удобство описания синтаксиса языка,
  + существование алгоритма, работающего с этим описанием языка с  
    достаточной скоростью.

Одним из способов описания синтаксиса языка (применимого и используемого и для формальных языков), допускающих использование эффективных алгоритмов, являются сетевые грамматики. Этот вид описания возник уже довольно давно, и постепенно все больше и больше совершенствовался и обобщался. Представление грамматики в виде графа отличается наглядностью и компактностью; правда, эти наглядность и компактность в сильной степени зависят от выбранного способа его описания.

Одним из простых вариантов описания некоторого языка является использование конечных автоматов. Этому способу посвящено большое количество литературы, в частности, он может использоваться для языков программирования и в исследовании операций в качестве очень удобного формального описания различных процессов. В то же время необходимо отметить важный недостаток такого описания языка: применяя его, можно установить принадлежность анализируемой цепочки символов языку, но ее структура для нас останется неизвестной, т. к. все символы для нас равноправны.

Следующим этапом развития сетевых грамматик можно считать рекурсивные сети переходов. В рамках этого подхода язык можно описать набором конечных автоматов. При этом к множеству терминальных символов каждого автомата добавляется и множество нетерминальных символов; каждый из них соответствует одному автомату. Если при разборе входной цепочки анализатор встречает в графе нетерминальный символ, то он запоминает состояние (это можно делать в стеке) и переходит к анализу по тому конечному автомату, который соответствует обнаруженному нетерминальному символу. После окончания анализа в рамках этого автомата анализатор возвращается на предыдущий уровень. При этом возможна рекурсия, как прямая, так и косвенная. Среди нетерминальных символов должен быть один выделенный, с которого и начинается анализ.[12]

Таким образом, кроме качественного ответа, принадлежит ли данная цепочка символов языку или нет, анализатор может возвратить и структуру этой цепочки, например, в виде дерева, где корневой вершине соответствует начальный конечный автомат, вершины следующего, дочернего уровня представлены последовательностью разобранных терминальных символов (которым соответствуют «листья» этого дерева) и нетерминальных символов, которые являются корнями поддеревьев, аналогичных главному.[12]

В общем случае из-за недетерминированности выбора следующей вершины на каждом этапе анализа (если из текущей вершины ведет несколько дуг) в результате может получиться много вариантов структуры. Вообще типичная проблема, возникающая (или грозящая возникнуть) - это опасность комбинаторного взрыва.

Обобщением рекурсивной сети переходов является предложенная Вудсом модель расширенной сети переходов. Она предназначена для контроля контекстных условий и позволяет существенно увеличить множество анализируемых языков. Обобщение осуществляется следующим образом. Каждой дуге (соединяющей два последовательно разбираемых терминальных или нетерминальных символа) можно поставить в соответствие операторы, определяющие возможность перехода по дуге. Их удобно представлять в виде булевских функций (если такой оператор возвращает значение истина, то это будем интерпретировать как разрешение перехода по дуге, которой приписан оператор, а если ложь, то как запрет). Операторы, приписанные одной дуге, можно считать соединенными логической связкой и; таким образом, переход по дуге осуществляется, если все ее операторы позволяют совершить этот переход.

Кроме операторов, соответствующих дугам, удобно ввести операторы, выполняющиеся после окончания работы конечного автомата при переходе на предшествующий уровень вложенности. Если их конъюнкция примет значение ложь, считается, что анализ, осуществляемый этим конечным автоматом, неуспешен. [12]

Смысл деятельности операторов может заключаться не только в возвращаемом значении, но и в т. н. побочном эффекте: они могут сохранять некоторые факты в специально отведенной памяти для других операторов, которые могут быть вызваны позднее. Каждый оператор имеет доступ к информации о разобранном (или «свернутом») нетерминале, из которого ведет дуга, и к информации, общей для активного в этот момент времени конечного автомата. [12]

Отметим следующие недостатки описания РСП:

* + "Недетализированность" правил. Каждая синтаксическая связь  
    конкретных словоформ имеет свои особенности, отличия от  
    общей схемы, описанной в грамматике.
  + Фактическая невозможность использования частотных или  
    вероятностных оценок, поскольку такие оценки можно приписать  
    только правилам переходов по дугам. Таким образом, РСП-  
    анализатор выдает при анализе предложения целый набор  
    возможных вариантов, но возможных только с его точки зрения,  
    точнее было бы сказать - формально правильных. Почти всегда  
    есть один наиболее приоритетный вариант, который и требуется  
    получить в первую очередь, остальные следует отбросить, и  
    именно здесь нам требуются частотные оценки.
  + Невозможность объединить описание синтаксиса и семантики,  
    поскольку не существует "семантической" грамматики. [12]

**Алгоритм исправления ошибок.**

Для описания алгоритма необходимо дать несколько определений.

**Определение:** Отображение ошибки категории z = {z1, z2, z3, z4} данной словоформы – множество словоформ, порождаемых этой словоформой в результате всех возможных ошибок категории zi.

**Определение:** Полное отображение одиночной ошибки данной словоформы – множество словоформ, порождаемых этой словоформой в результате всех возможных ошибок четырех категорий.

Таким образом, если через Г1 (словоформа) обозначить множество словоформ, порождаемых данной словоформой в результате ошибки категории 1 (отображение ошибки категории 1), через Г2 – отображение ошибки категории 2, через Г3 – отображение ошибки категории 3 и через Г4 – отображение ошибки категории 4, то полное отображение одиночной ошибки Г (словоформа) есть объединение всех четырех множеств [4].

**Определение:** Полное обратное отображение одиночной ошибки данной словоформы – множество словоформ, порождающих данную словоформу в результате одиночной ошибки каждой из четырех категорий [4].

Описание метода полных обратных преобразований:

1. Вносим в буквенную цепочку полное обратное преобразование.
2. Каждый из получившихся токенов проверяем на наличие в словаре.
3. Если токен имеется в словаре, то он добавляется в список корректных кандидатов.

Таким образом обеспечивается 100% вероятность коррекции ошибок, если корректное слово имеется в словаре. Однако данный алгоритм обладает неустранимой особенностью – все обрабатываемые токены никак не оцениваются, а потому невозможно выбрать наиболее подходящий вариант для исправления ошибки, т.е. требуется вмешательство оператора.

Для успешного процесса коррекции важны эффективные алгоритмы диагностики орфографических ошибок. Таким образом процесс диагностики ошибок в словах сводится к определению принадлежности последовательности букв (токена) к данному естественному языку [3].

Таким образом, исправление ошибок определенных классов, в том числе однобуквенных, является практически важной задачей. Алгоритмы исправления ошибок в русских словах должны учитывать особенности русского языка как высоко флективного [3].

**Исследование и построение решения задачи.**

При анализе литературы проанализированы некоторые алгоритмы диагностики и коррекции одиночных ошибок в текстах естественного языка, их достоинства и недостатки. На данный момент имеются собственные реализации, что позволило сделать сравнительный анализ.

На каждом из алгоритмов было проведено два варианта тестирования: диагностика и коррекция. Тестирование коррекции проводилось в два этапа: тестирование коррекции изолированных слов с помощью алгоритма, основанного на морфологическом анализе и тестирование коррекции слов в окружении приложения на основе статистики встречаемости словосочетаний в корпусах текстов.

Тестирование проводилось следующим образом. Бралось заведомо корректное слово из словаря. Получалось его полное обратное отображение. Каждое из получившихся слов подавалось на вход алгоритма коррекции ошибок. На выходе должно было получится слово идентичное начальному, заведомо корректному. Схема представлена на Рисунке №1.

Словарная

статья

Полное

обратное

преобразование

Коррекция

Варианты

коррекции

Сравнение

Результат

Рисунок №1. Схема тестирования алгоритмов.

**Метод n-грамм.**

Данный алгоритм был выбран из-за высокой скорости работы, простоты реализации и популярности. Для алгоритма n-грамм был построен словарь на основе словаря [11]. 3028433 слова были представлены в виде списка, потом разбиты на всевозможные n-граммы. В результате было получено 964 2-грамм, 13400 3-грамм, 89093 4-грамм. Далее эти n-граммы использовались в тестировании алгоритма.

Описание алгоритма:

1. Пусть имеется словарь из n-грамм.
2. Устанавливаем указатель в начало слова.
3. Считываем n символов после указателя.
4. Проверяем наличие n-граммы в словаре.
5. Сдвигаем указатель на один символ, переходим на шаг 3.

В результате при тестировании возможностей диагностики алгоритму n-грамм было дано 595192 последовательности символов. В результате 247872 последовательности символов продиагностировано неверно. Т.е. алгоритм n-грамм вернул утвердительный результат – слова принадлежат словарю русского языка, тогда как в реальности эти слова являлись ошибочными. Надо заметить, что словарь n-грамм составлен избыточно, т.к. при наличии 3-грамм в словаре, наличие 2-грамм необязательно. Они являются подмножеством 3-грамм.

Коррекция проводилась на меньшем количестве слов. На вход подавалось 10 заведомо корректных слов, после полного обратного преобразования – 2700 заведомо некорректных. Алгоритм n-грамм затратил за коррекцию каждого слова 0.017 секунды. При этом на каждое неправильное слово было возвращено около 250 вариантов слов. Однако, не все эти слова на самом деле принадлежали словарю русского языка.

**Метод, основанный на морфологическом анализе.**

Как показывают исследования, метод n-грамм не дает даже 50% гарантию исправления ошибки в слове, поэтому был исследован алгоритм диагностики ошибок, основанный на морфологическом анализе.

Описание реализованного алгоритма (листинг в Приложении B):

1. Проводим всевозможные разбиения слова на префикс и остальную часть. Проверяем наличие префикса в словаре. Если его там нет, значит либо в нем имеется ошибка, либо разбиение оказалось неверным. Для каждого префикса находим полное обратное преобразование. Каждый из полученных токенов проверяем на наличие в словаре. Если токен там присутствует, значит ошибка исправлена (конец алгоритма). Если префикс в словаре имеется, значит ошибка присутствует в остальной части слова (переход в 2). Если проведены всевозможные разбиения и ни одного токена в словаре не найдено, значит слова в словаре нет или в нем более одной ошибки.
2. Производим всевозможные разбиения оставшейся части слова на корень и постфикс. Если корень найден в словаре и он совместим с префиксом, значит ошибка в постфиксе (переход в 3). Если корень в словаре не найден, находим полное обратное преобразование. Каждый из полученных токенов проверяется на наличие в словаре. Если токен найден и совместим имеющимся префиксом и постфиксом, значит ошибка исправлена (конец алгоритма). Если проведены всевозможные разбиения и ни одного токена в словаре не найдено, значит слова в словаре нет или в нем более одной ошибки.
3. Проводим поиск постфикса в словаре. Если он найден, значит ошибки в слове нет. Если не найден, значит находим полное обратное преобразование и каждый полученный токен ищем в словаре. Если он найден и совместим с имеющимся корнем и префиксом, значит ошибка исправлена, если нет, значит в слове более двух ошибок. Если проведены всевозможные разбиения и ни одного токена в словаре не найдено, значит слова в словаре нет или в нем более одной ошибки.

При выборе структуры словаря были рассмотрены модели русского языка [3], а так же учитывались рекомендации [9].

Общим подходом словарь похож на корневую часть словаря [5] и представляет собой текстовый файл в особом формате. Он содержит примерно 143702 корней и 2674 модели, что позволяет покрыть достаточно большую часть русского языка (3028433 словоформ). Первая секция представляет набор моделей. Моделью называется совокупность пар префикса, идентификатора модели и постфикса разделенных символом ‘**\***’. Вторая секция представляет набор корней с идентификаторами модели.

Пример:

%НЫЙ\*йа%НОГО\*йб%НОМУ\*йв%НОГО\*йг%НЫЙ\*Рщ ….

ИНТЕРЕС 19

Таким образом достигается хороший процент сжатия словаря по сравнению с простым перечислением словоформ.

Описанный в [3] алгоритм основан на префиксальной части словаря [5], потому перед его использованием проводилась процедура преобразования из одного формата в другой.

В начале реализации метода коррекции орфографических ошибок на основе морфологического анализа была использована реляционная база данных (на основе СУБД SQLite) для хранения словаря. Однако проведенные нами тесты показали, что более 90% процессорного времени тратиться на разбор sql-выражений, поэтому было принято решение отказаться от использования реляционных БД. В результате была разработана структура для хранения словаря на основе конечного автомата, для хранения буквенных цепочек в виде дерева. Структура состоит из автомата, распознающего приставки, и ссылок на группу автоматов, распознающих корни, совместимые с приставками. Тоже самое представляет собой автомат корней. Структуру завершает автоматы окончаний. Таким образом, структура словаря представляет дерево автоматов.

Создание специализированного хранилища для хранения словаря позволило увеличить производительность и уменьшить сложность выборки слова для работы алгоритма коррекции орфографических ошибок с помощью морфологического анализа.

При тестировании алгоритма описанного в [10] были допущены некоторые неточности, повлиявшие на достоверность результатов:

1. В словарь входили словоформы только тех слов, которые присутствовали в тексте, т.е. бралось слово из текста, приводилось к неопределенной форме, брались все его словоформы и вносились в словарь. Таким образом, авторы работы [10] исключили возможные неоднозначности коррекции ошибок. Например, если на вход алгоритма коррекции будет подано слово *прйложить*, и словаре будут присутствовать слова *преложить* и *приложить*, то они будут выданы в качестве вариантов коррекции ошибок. А по результатам тестирования [10] неточность коррекции могла быть только в случаях неоднозначности окончаний.
2. В работе [10] при имитации появления ошибок в тексте было принято, что ошибки разных типов имеют одинаковую вероятность появления, что противоречит статистике приведенной в [10].

При исследовании алгоритмов реализованных в данной работе эти неточности были учтены.

Алгоритм диагностики ошибок, основанный на морфологическом, анализе показал 100% успешность диагностики 595192 ошибочных словоформ. Т.е. во всех случаях был возвращен отрицательный результат – слова не принадлежат словарю русского языка.

При исследовании метода морфологического анализа при коррекции ошибок в словах на каждое заведомо неверное слово алгоритм возвращал список слов, состоящий из 5-10 вариантов исправления ошибки. При этом каждое из этих слов присутствовало в словаре и в этом списке обязательно присутствовало начальное, верное слово. Время коррекции одного слова составило примерно 0.54 секунды.



Рисунок №2. Сравнительные результаты тестирования диагностики методов, основанных на n-граммах и морфологическом анализе.

**Метод сокращения вариантов исправления ошибок в словах.**

При использовании метода диагностики и коррекции ошибок в словах на основе морфологического анализатора возникает проблема неоднозначного словарного сокращении вариантов выдачи алгоритма основанного на морфологическом анализаторе. В качестве вариантов решения данной проблемы были рассмотрены методы основанные на частотных словарях и синтаксических анализаторах. Использование адекватного синтаксического анализатора в данной работе не представляется возможности из-за крайней сложности и малой эффективности относительно простых алгоритмов. Таким образом, нами было принято решение использовать методики, основанные на основе частотных словарей.

Проанализировано 677 художественных текстов и собрана информация о 17272996 словосочетаний по 2 и 3 слова. Процесс анализа словосочетаний включает в себя фрагментацию текста и сбор статистики встречаемости словосочетаний. В процессе анализа текстов было выявлена весьма низкая производительность СУБД при выполнении запросов INSERT, поэтому было принято решение о переносе всей логики выполнения sql-запросов в БД посредством выполнения хранимых процедур.

Непосредственно в базе данных хранится информация об обработанных словоформах и словосочетаниях. Например, если в тексте словосочетания «они ходят» и «они собирают грибы» встретились 2 и 5 раз соответственно, то Таблицы №1, №2, №3 будут содержать следующую информацию.

|  |  |
| --- | --- |
| Id | Wordform |
| 1 | Они |
| 2 | Собирают |
| 3 | Грибы |
| 4 | Ходят |

Таблица №1. Wordforms.

|  |  |
| --- | --- |
| Word | Phrase |
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 1 | 2 |
| 4 | 2 |

Таблица №2. WordToPhrases.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Word1 | Word2 | Word3 | Count |
| 1 | 1 | 2 | 3 | 2 |
| 2 | 1 | 4 | NULL | 5 |

Таблица №3. Phrases.

Хранимые процедуры:

1. Добавление фразы / обновление статистики.
   1. Выборка индексов слов посредством вызова процедуры addword.
   2. Выборка индекса словосочетания с помощью индексов слов.
   3. Если словосочетание присутствует, тогда данные о его встречаемости обновляются, иначе словосочетание добавляется с первоначальными данными.
2. Добавление словоформы.
   * 1. Производится выборка словоформы.
     2. Если она отсутствует в словаре, тогда производится добавление.
     3. Индекс словоформы возвращается в качестве результата.
3. Выборка встречаемости по словосочетанию.
   * + 1. Производится выборка встречаемости по словам сходящим в словосочетание.
       2. Если результат не определен, возвращается нуль.

Листинги хранимых процедуры приведены в Приложении С.

Таким образом была достигнута достаточная скорость работы СУБД Postgresql под управлением OC Windows 2003 Server – в среднем 3 часа на обработку одного текстового файла.

Перед непосредственной обработкой текста применялся **алгоритм фрагментации**:

1. Все символы в тексте приводятся к нижнему регистру.
2. В тексте производится замена по символам перевод на новую строку, возврат каретки к началу строки на пробел.
3. Далее текст делится на группы токенов по символам точка, запятая, точка с запятой, тире, двоеточие, вертикальные и угловые двойные кавычки, открывающая и закрывающая круглые скобки.
4. Каждая группа токенов делится по пробелам на токены и формируются словосочетания по 2 и 3 слова.
5. Считается встречаемость каждого словосочетания в тексте и в конце данные заносятся в БД.

**Алгоритм комплексной проверки:**

1. Корректируем каждое слово в группе посредством алгоритма на основе морфологического анализа.
2. С каждым вариантом коррекции ошибочного слова формируются образцы словосочетаний по 2 и 3 слова.
3. По каждому образцу делается запрос в БД об их встречаемости.
4. Выбирается образец с максимальной частотностью и делается замена в исходном предложении.

Тестирование алгоритма коррекции слов в окружении предложения на основе статистики встречаемости словосочетаний в корпусах текстов проводилось следующим образом. Перебирались все слова корректного предложения. В каждое слово вносилась одна из четырех типов ошибок и получалось полное обратное преобразование. Далее слово с ошибкой вставлялось обратно в

Входной текст

Фрагментация

Формирование

вариантов

коррекции

Подстановка

вариантов

коррекции в

словосочетание

с некорректным

словом

Подстановка

результата

Выходной текст

Рисунок №3. Схема улучшенного алгоритма диагностики и коррекции на основе морфологического анализа.

предложение. Получившееся предложение проверялось алгоритмом коррекции слов в окружении предложения на основе статистики встречаемости словосочетаний в корпусах текстов. Результаты тестирования представлены в Таблице №4 и Рисунке №4.

|  |  |  |  |
| --- | --- | --- | --- |
| Некорректные предложения | Варианты исправления | Корректные варианты исправления | % |
| 441 | 458 | 458 | 100 |
| 644 | 140 | 138 | 98 |
| 100 | 267 | 174 | 65 |
| 508 | 235 | 137 | 58 |
| 780 | 311 | 276 | 88 |

Таблица №4. Данные тестирования алгоритма на основе частотного словаря.



Рисунок №4. Сравнительные результаты тестирования коррекции методов, основанных на n-граммах и морфологическом анализе.

**Описание практической части.**

В реализации библиотеки был использован алгоритм описанный выше, что позволило добиться относительно неплохой производительности. Был использован компактный словарь [11], хотя в работе и не ставилась задача оптимизации по занимаемому дисковому пространству, размер словаря составил около 5 Мб. Данный словарь содержит около 173 780 основами (более чем 3 000 000 слов). В качестве структуры хранения было использовано дерево конечных автоматов, что позволило увеличить быстродействие на 2 порядка по сравнению с использованием реляционных БД. Реализация выполнена на языке С++ с использованием переносимых библиотек, потому библиотека может быть использована на платформах Linux, Windows, MacOS X. База данных статистики словосочетаний слов в корпусах текстов организована на основе СУБД PostgreSQL 8.2. Проанализировано 677 художественных текстов и собрана информация о 17272996 словосочетаний по 2 и 3 слова. Размер БД составил приблизительно 5150 Мб. Реализация алгоритма коррекции слов на основе статистики встречаемости слов в корпусах текстов написана на языке C# платформа MS.Net 2.0.

**Выводы.**

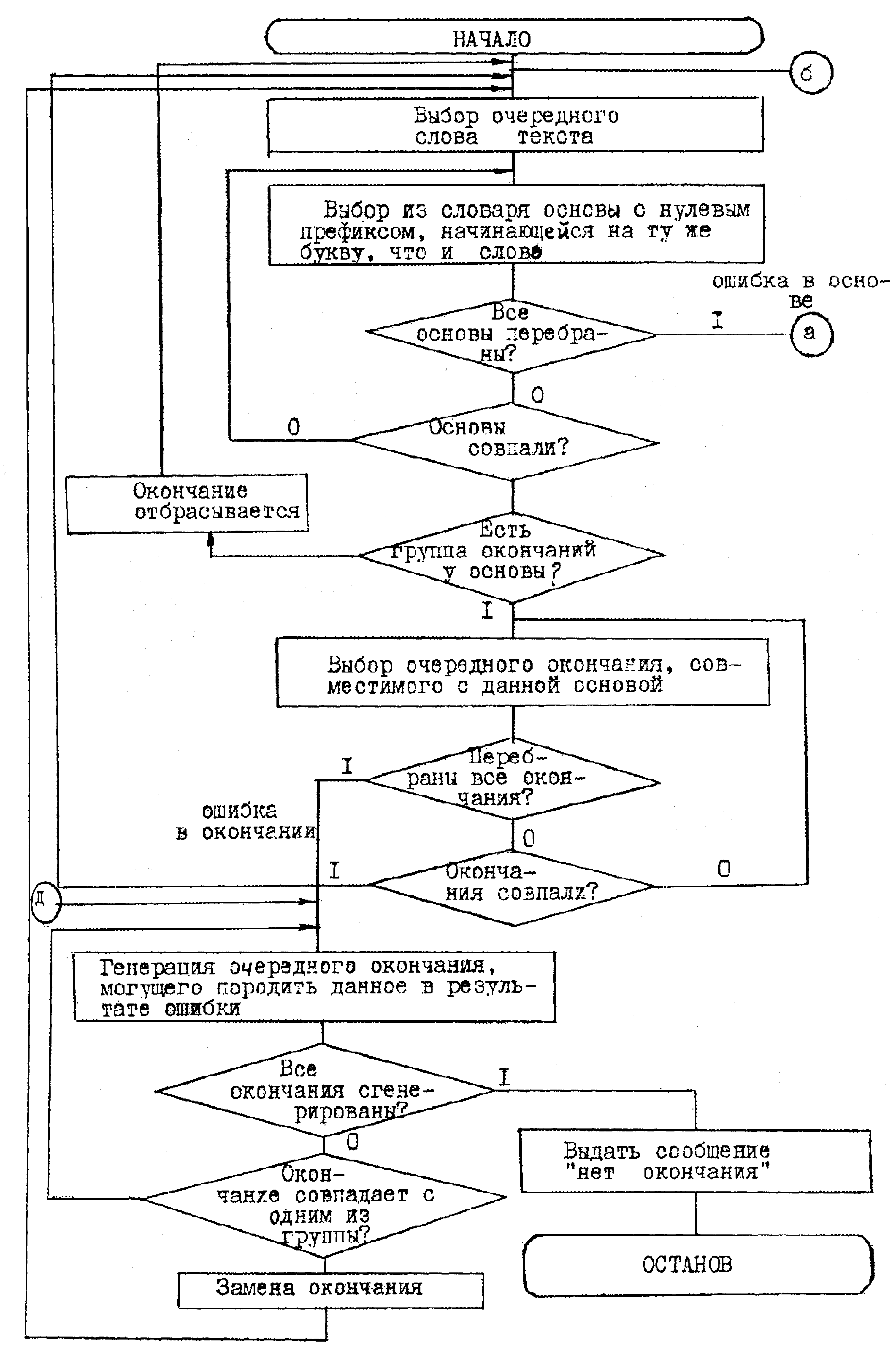
В данной работе рассмотрена проблема автоматической диагностики и коррекции одиночных ошибок в текстах русского языка. Основные результаты состоят в следующем:

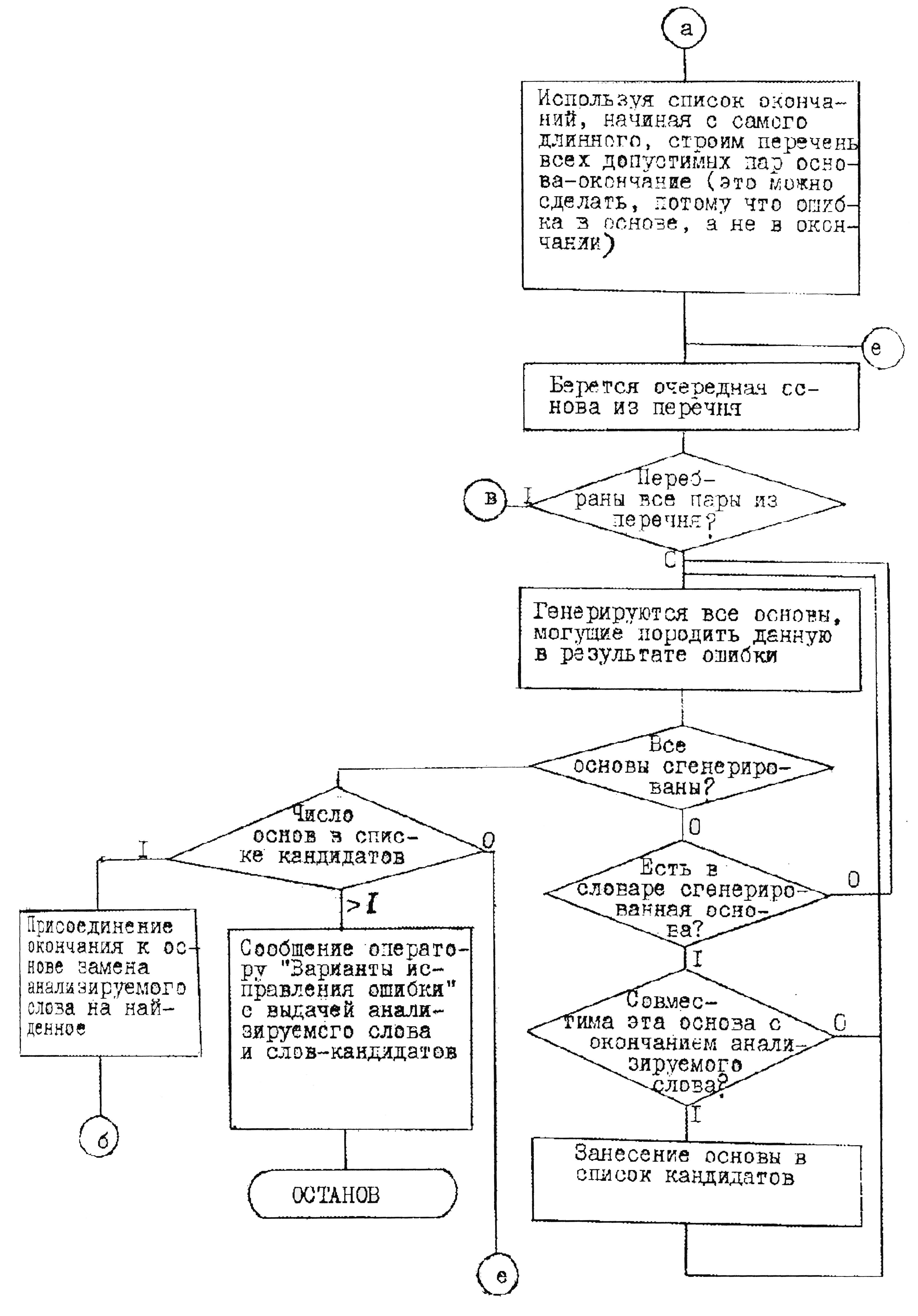
1. Проведен сравнительный анализ методов диагностики и коррекции одиночных орфографических ошибок в текстах русского языка. Выявлено что системы учитывающие особенности языка лучше справляются с поставленной задачей.
2. Уменьшено количество вариантов коррекции в методе, основанном на морфологическом анализе, за счет применения частотного словаря, что позволило повысить процент корректно исправленных слов в 4 раза.

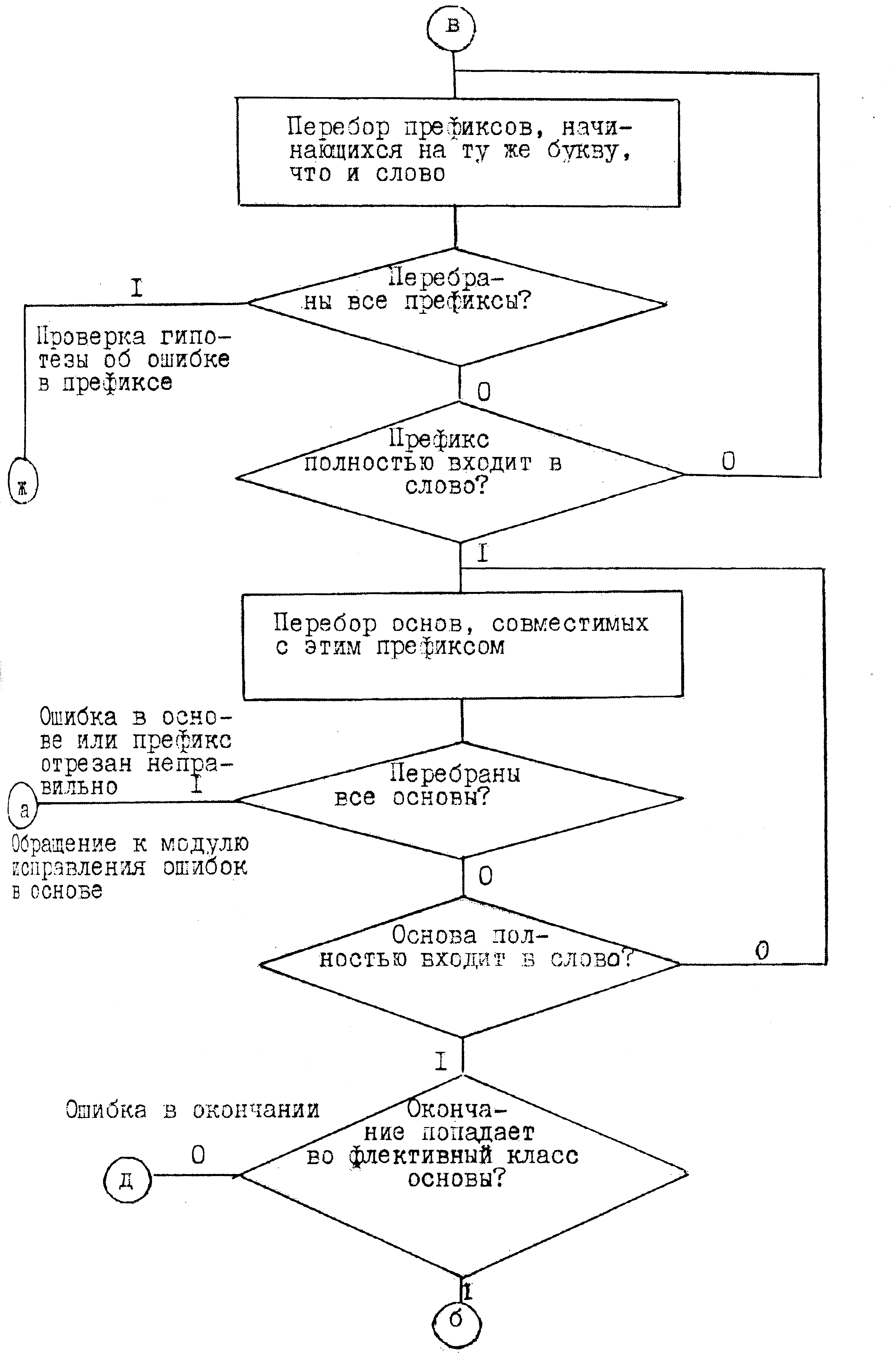
**Список литературы.**

1. Гниловская Л.П. Гниловская Н.Ф. Автоматическая коррекция орфографических ошибок // Культура народов Причерноморья. – 2004. - № 48. – с. 171 – 180.
2. Бутакова Л.О. Опыт классификации ошибок, свойственной письменной речи. 1998. Internet: <http://www.omsu.omskreg.ru/vestnik/articles/y1998-i2/a072/article.html>
3. Кондратюк Д.Л. Разработка и исследование метода повышения надежности вычислительных устройств, реализующих эвристические алгоритмы. – Севастополь, 1993. – 224 с.
4. Зализняк А.А. Грамматический словарь русского языка: Словоизменение. – М.: Рус. яз., 1987. – 880 с.
5. Кузнецова А.И., Ефремова Т.Ф. Словарь морфем русского языка. – М: Русский язык, 1986. – 1134 с.
6. Полюга Л.М. Морфемній словник.– К.: Рад. школа, 1983. – 464 с.
7. Морфема. Википедия, 2007. Internet: <http://ru.wikipedia.org/wiki/Морфема>.
8. Прохоров А.М. Советский энциклопедический словарь. – М.: Сов. энциклопедия, 1989. – 1632 с.
9. Ножов И.М. Морфологическая и синтаксическая обработка текста (модели программы). – Москва, 2003. – 140 с.
10. Гельбух А.Ф. Эффективно реализуемая модель морфологии флективного естественного языка. – Москва, 1994. – 77 с.
11. Русский морфологический словарь. АОТ. Internet: [http://www.aot.ru](http://www.aot.ru/)
12. Одинцев Н.В. Адаптивный синтаксический анализатор русского языка. – Москва, 2003. – 101 с.
13. Мазов Н.А. N-граммные методы обработки текстовой информации. 1995. Internet: <http://www.gpntb.ru/win/inter-events/crimea95/report/rep075_r.html>
14. Spevack M. A Complete and systematic concordance to the works of Shakespeare. Hildesheim, 1968-70.
15. Словарь языка Пушкина, М., 1956-61.
16. Punto Switcher. Яндекс. Internet: http://punto.ru.

**Приложение А.**







Приложение В.

Листинг №1. Исходный код реализации алгоритмов коррекции слова с помощью морфологического анализа.

**namespace** NSWords

{

// коррекция слова

std::vector< std::string >

CorrectorGramError::CorrectionWord(**const** std::string &word)

{

std::vector<std::string> correctWords;

**if** (Dict.IsFindWord(word))

{

correctWords.push\_back(word);

}

**else** // значит есть ошибка в слове

{

// Пусть в слове нет приставки

std::vector<std::string> tmp(CRP("", word));

correctWords.insert(

correctWords.end(), tmp.begin(), tmp.end());

// Значит в слове есть приставка

**if** (correctWords.size() == 0)

{

tmp = CPRP(word);

correctWords.insert(

correctWords.end(),

tmp.begin(),

tmp.end());

}

}

**return** correctWords;

}

// корректировка корня или окончания,

// при условии что префикс верный

std::vector<std::string>

CorrectorGramError::CRP(

const std::string &prefix,

const std::string &word )

{

std::vector<std::string> correctWords;

std::map<std::string, int> tmpCorrectWords;

**for** (int i = 0; i < word.length(); i++)

{

// разбиваем слово на основу и окончание

std::string root(word.substr(0, i + 1));

std::string postfix(word.substr(i + 1));

// ищем ошибку в корне посредством перебора

// результата полного обратного перебора.

BOOST\_FOREACH (std::string tmpRoot, BFT(root))

{

BOOST\_FOREACH (Lemma lemma, Dict.FindRoot(tmpRoot))

{

**if** (lemma.Model.IsFindPrefixPostfix(

prefix, postfix))

{

TmpCorrectWords[prefix + tmpRoot + postfix]++;

}

}

}

std::vector<std::string> bftPostfix = BFT(postfix);

BOOST\_FOREACH (Lemma lemma, Dict.FindRoot(root))

{

BOOST\_FOREACH (std::string tmpPostfix, bftPostfix )

{

**if** ( lemma.Model.IsFindPrefixPostfix(

prefix, tmpPostfix))

{

tmpCorrectWords[prefix + root + tmpPostfix]++;

}

}

}

}

std::map< std::string, int >::iterator pos;

**for** (pos = tmpCorrectWords.begin();

pos != tmpCorrectWords.end(); pos++)

{

correctWords.push\_back( pos->first );

}

**return** correctWords;

}

// корректировка префикса

std::vector<std::string>

CorrectorGramError::CPRP(const std::string &word)

{

std::vector<std::string> сorrectWords;

// пусть в слове приставка верная.

**for** (int i = 0; i < word.length(); i++)

{

// разбиваем слово на основу и окончание

std::string prefix(word.substr(0, i + 1));

std::string other(word.substr(i + 1));

BOOST\_FOREACH (const Model &model, Dict.FindPrefix(prefix))

{

std::vector<std::string> tmp(CRP(prefix, other));

correctWords.insert(

correctWords.end(),

tmp.begin(),

tmp.end() );

}

}

// пусть приставка в слове неверная

**if** (correctWords.size() == 0)

{

**for** (int i = 0; i < word.length(); i++)

{

// разбиваем слово на основу и окончание

std::string prefix(word.substr(0, i + 1));

std::string other(word.substr(i + 1));

BOOST\_FOREACH (std::string tmpPrefix, BFT(prefix))

{

BOOST\_FOREACH (

**const** Model &model,

Dict.FindPrefix(tmpPrefix))

{

std::vector<std::string> tmp(

CRP(tmpPrefix, other));

correctWords.insert(

correctWords.end(),

tmp.begin(),

tmp.end());

}

}

}

}

**return** correctWords;

}

Листинг №2. Исходный код алгоритма фрагментации текста.

(C# MS.Net 2.0)

**public** IList<IList<**string**>> Fragmentation(**string** text)

{

IList<IList<**string**>> result = **new** List<IList<**string**>>();

Regex re = **new** Regex("\"|«|»|\t|\n|\r");

Regex re1 = **new** Regex(@"(\w+)");

**foreach** (**string** s **in** re.Replace(text, " ").ToLower().Split(

**new** char[] {'.', ',', ';', '-', ':', '(', ')'}))

{

IList<**string**> words = **new** List<**string**>();

MatchCollection re1\_result = re1.Matches(s);

**for** (int i = 0; i < re1\_result.Count; i++)

{

words.Add(re1\_result[i].Value);

}

result.Add(words);

}

**return** result;

}

Листинг №3. Исходный код алгоритма сокращения вариантов коррекции. (C# MS.Net 2.0)

**private** IList<**string**> CorrectSentence(**string** sentence)

{

IList<**string**> correctSentences = **new** List<**string**>();

IList<IList<**string**>> words = Fragmentation(sentence);

**int** word\_num = 1;

CorrectorGramError gramCorrector = **new** CorrectorGramError();

**foreach** (IList<**string**> words\_group **in** words)

{

**for** (**int** i = 0; i < words\_group.Count; i++)

{

IList<**string**> correctWords = gramCorrector.CorrectionWord(words\_group[i]);

**if** (correctWords.Count > 1)

{

IList<IList<**string**>> phrases =

**new** List<IList<**string**>>();

IDictionary<IList<**string**>, **int**> correctPhrases =

**new** Dictionary<IList<**string**>, **int**>();

IList<**string**> phrase;

// -- обработка двойных словосочетаний --

**if** (i + 1 < words\_group.Count)

{

phrase = **new** List<**string**>();

phrase.Add(words\_group[i]);

phrase.Add(words\_group[i + 1]);

phrases.Add(phrase);

**foreach** (

KeyValuePair<IList<**string**>, **int**> pair **in** CorrectPhrase(

phrase,

correctWords,

words\_group[i]))

{

correctSentences.Add(pair.Key[0]);

}

}

**if** (i - 1 >= 0)

{

phrase = **new** List<**string**>();

phrase.Add(words\_group[i - 1]);

phrase.Add(words\_group[i]);

phrases.Add(phrase);

**foreach** (

KeyValuePair<IList<**string**>, **int**> pair **in**

CorrectPhrase(

phrase,

correctWords, words\_group[i]))

{

correctSentences.Add(pair.Key[1]);

}

}

// --------------------------------------

// -- обработка тройных словосочетаний --

**if** (

(i - 1 >= 0) &&

(i + 1 < words\_group.Count))

{

phrase = **new** List<**string**>();

phrase.Add(words\_group[i - 1]);

phrase.Add(words\_group[i]);

phrase.Add(words\_group[i + 1]);

phrases.Add(phrase);

**foreach** (

KeyValuePair<IList<**string**>, **int**> pair **in**

CorrectPhrase(

phrase,

correctWords, words\_group[i]))

{

correctSentences.Add(pair.Key[1]);

}

}

**if** (i + 2 < words\_group.Count)

{

phrase = **new** List<**string**>();

phrase.Add(words\_group[i]);

phrase.Add(words\_group[i + 1]);

phrase.Add(words\_group[i + 2]);

phrases.Add(phrase);

**foreach** (

KeyValuePair<IList<**string**>, **int**> pair **in**

CorrectPhrase(

phrase,

correctWords, words\_group[i]))

{

correctSentences.Add(pair.Key[0]);

}

}

**if** (i - 2 >= 0)

{

phrase = **new** List<**string**>();

phrase.Add(words\_group[i - 2]);

phrase.Add(words\_group[i - 1]);

phrase.Add(words\_group[i]);

phrases.Add(phrase);

**foreach** (

KeyValuePair<IList<**string**>, **int**> pair **in**

CorrectPhrase(

phrase,

correctWords, words\_group[i]))

{

correctSentences.Add(pair.Key[2]);

}

}

// --------------------------------------

}

}

}

**return** correctSentences;

}

**private** IDictionary<IList<**string**>, **int**>

CorrectPhrase(

IList<**string**> phras,

IList<string> correctWords, **string** wg)

{

IDictionary<IList<**string**>, **int**> correctPhrases = **new** Dictionary<IList<**string**>, **int**>();

**foreach** (**string** word **in** correctWords)

{

IList<**string**> phrasetmp = **new** List<**string**>();

**for** (**int** j = 0; j < phras.Count; j++)

{

**if** (phras[j] == wg) /\*words\_group[i]\*/

{

phrasetmp.Add(word);

}

**else**

{

phrasetmp.Add(phras[j]);

}

}

**using** (NpgsqlCommand cmd = **new** NpgsqlCommand())

{

cmd.Connection = sql\_conn;

**if** (phrasetmp.Count == 3)

{

cmd.CommandText =

String.Format(

"SELECT GetPhraseStat('{0}', '{1}', '{2}')",

phrasetmp[0],

phrasetmp[1],

phrasetmp[2]);

}

**else**

{

cmd.CommandText =

String.Format(

"SELECT GetPhraseStat('{0}', '{1}', NULL)", phrasetmp[0],

phrasetmp[1]);

}

cmd.CommandTimeout = 80000;

**int** phraseStat = (**int**)cmd.ExecuteScalar();

**if** (phraseStat > 0)

{

correctPhrases.Add(phrasetmp, phraseStat);

}

}

}

**return** correctPhrases;

}

**Приложение С.**

1. **Листинг хранимой процедуры выборки встречаемости по словосочетанию:**

**CREATE OR REPLACE FUNCTION** getphrasestat(**character varying**, **character varying**, **character varying**)

**RETURNS integer** **AS**

**$BODY$**

**DECLARE**

phrase\_count **integer**;

**BEGIN**

**IF** $3 **IS NOT NULL**

**THEN**

**SELECT**

count **INTO** phrase\_count

**FROM**

phrases

**WHERE**

word1 = (**SELECT** wordform\_id **FROM** wordsforms **WHERE** wordform = $1)

**AND**

word2 = (**SELECT** wordform\_id **FROM** wordsforms **WHERE** wordform = $2)

**AND**

word3 = (**SELECT** wordform\_id **FROM** wordsforms **WHERE** wordform = $3);

**ELSE**

**SELECT**

count **INTO** phrase\_count

**FROM**

phrases

**WHERE**

word1 = (**SELECT** wordform\_id **FROM** wordsforms **WHERE** wordform = $1)

**AND**

word2 = (**SELECT** wordform\_id **FROM** wordsforms **WHERE** wordform = $2)

**AND**

word3 **IS** **NULL**;

**END IF**;

**IF** phrase\_count **IS** **NULL**

**THEN**

phrase\_count = 0;

**END** **IF**;

**RETURN** phrase\_count;

**END;**

**$BODY$**

**LANGUAG**E 'plpgsql' **VOLATILE**;

1. **Листинг хранимой процедуры добавления словоформы:**

**СREATE** **OR** **REPLACE** **FUNCTION** addword(**text**)

**RETURNS integer AS**

**$BODY$**

**DECLARE**

word\_id **integer**;

**BEGIN**

-- выбираем идентификатор слова.

**SELECT** wordform\_id **INTO** word\_id **FROM** wordsforms **WHERE** wordform = $1;

**IF** word\_id **IS NULL**

**THEN**

**INSERT INTO** wordsforms (wordform) **VALUES** ( $1 ) **RETURNING** wordform\_id **INTO** word\_id;

**END IF**;

**RETURN** word\_id;

**END;**

**$BODY$**

**LANGUAGE** 'plpgsql' **VOLATILE**;

1. **Листинг хранимой процедуры добавления/обновления статистики встречаемости словосочетания.**

**CREATE OR REPLACE FUNCTION** addphrase(**character varying**, **character varying**, **character varying**, num **integer**)

**RETURNS** integer **AS**

**$BODY$**

**DECLARE**

word1\_id **integer**;

word2\_id **integer**;

word3\_id **integer**;

phraseId **integer**;

**BEGIN**

word1\_id := addword($1);

word2\_id := addword($2);

**IF** $3 **IS NOT NULL THEN** word3\_id := addword($3); **END IF**;

**IF** $3 **IS NOT NULL**

**THEN**

**SELECT** phrase\_id **INTO** phraseId **FROM** Phrases **WHERE** Word1 = word1\_id **AND** word2 = word2\_id **AND** word3 = word3\_id;

**ELSE**

**SELECT** phrase\_id **INTO** phraseId **FROM** Phrases **WHERE** Word1 = word1\_id **and** word2 = word2\_id **and** word3 **IS NULL**;

**END IF**;

**IF** phraseId **IS NULL**

**THEN**

**INSERT INTO** Phrases(word1, word2, word3, count) **VALUES** (word1\_id, word2\_id, word3\_id, num) **RETURNING** phrase\_id **INTO** phraseId;

**INSERT INTO** wordtophrases (word, phrase) **VALUES** (word1\_id, phraseId);

**INSERT INTO** wordtophrases (word, phrase) **VALUES** (word2\_id, phraseId);

**IF** $3 **IS NOT NULL**

**THEN**

**INSERT INTO** wordtophrases (word, phrase) **VALUES** (word3\_id, phraseId);

**END IF**;

**ELSE**

**UPDATE** Phrases **SET** Count = Count + num **WHERE** phrase\_id = phraseId;

**END IF**;

**RETURN** phraseId;

**END**;

**$BODY$**

**LANGUAGE** 'plpgsql' **VOLATILE**;