10 класс

Проект к главе 6

«Технология компьютерной обработки текста»

Теоретический минимум

Основные понятия

Представление информации в виде текста остается до сих пор одним из наиболее универсальных и стало одним из первых доступных для обработки с помощью компьютера. Энциклопедический словарь дает такое определение понятию «текст»: **«Текст** – это упорядоченный набор слов, предназначенный для того, чтобы выразить некий смысл. В лингвистике термин используется в широком значении, включая в себя и устную речь».

Представление информации в виде текста при обработке с помощью компьютера близко к этому определению. Под «текстовым» понимают такое представление информации, при котором она представлена в виде записи слов (логических элементов) некоторого языка и доступна для чтения человеком.

Язык для такого представления характеризуется в первую очередь некоторым алфавитом, т.е. допустимым набором символов. Поскольку компьютер работает только с двоичным кодом, то для записи и обработки текста требуется взаимно и однозначно сопоставить символы и некоторые двоичные коды. Такое *правило сопоставления кодов и символов алфавита называется* ***кодировкой текста.***

Для русского языка существует несколько вариантов кодирования. Во-первых, это кодировки на основе стандарта ASCII: CP866 (DOS), KOI-8R (UNIX), CP1251(Windows) и MacCyr. Во вторых, это общий универсальный стандарт кодирования UNICODE. Как правило, программные средства поддерживают либо один из стандартов первой группы, либо один из группы и UNICODE. Как вы знаете, при необходимости преобразовать текст из одной кодировки в другую не очень сложно.

С точки зрения хранения, чаще всего для работы с текстом применяются ***текстовые файлы*** – самый простой, компактный и легко используемый с технической точки зрения способ хранения и передачи текста. ***Текстовый файл*** *представляет собой последовательность символов, разбитых на абзацы или строки.* Понятие «текстового файла» не предусматривает строго заданного формата или расширения. Тем не менее, помимо характерной для той или иной ОС таблицы кодировки, в текстовых файлах могут применяться три основных способа деления текста на строки (абзацы):

1. Windows (DOS) – символы «Возврат каретки»+«Перевод строки» (CR+LF);
2. Unix – символ «Перевод строки» (LF);
3. MacOs – символ «Возврат каретки» (CR).

Текстовые файлы применяются для самых различных целей и часто оказываются формой хранения данных, описанных сложными формальными языками. Эти файлы часто используются для записи конфигурации ПО, документирования, переноса данных, описания HTML или XML кода.

Анализ текста на естественном языке

Традиционно, когда мы говорим об обработке текста, мы подразумеваем его создание и оформление - например, в виде макета печатного издания, со всеми его особенностями. Это

важнейшее (и самое известное) применение возможностей компьютеров в этой области, но далеко не единственное. Не менее важны для нас самые разные средства, позволяющие автоматизировать и более интеллектуальные действия, в частности, извлечение из текста новой, более полезной информации, верификация и т.п.. Пример таких действий - анализ текста.

До тех пор, пока текст невелик, мы можем проанализировать его без применения специальных средств, прочитав и обдумав. Хотя даже в этом случае многое будет зависеть от того, насколько хорошо развито умение извлекать из текста полезные сведения. В том случае, когда необходимо сделать контент-анализ достаточно объемной книги (нескольких книг) или множества любых текстовых источников, такой анализ становится как минимум затруднительным для пользователя – на чтение и обдумывание нужно потратить немало времени. Работа такого рода в настоящее время становится обязательной составляющей любой учебной или профессиональной деятельности. Когда времени мало, а текста много, возникает необходимость в автоматизированной обработке текстовых источников.

Анализ текста на естественном языке – очень многогранная задача. Различные методы и средства анализа разработаны для многих узких областей – это задачи поиска, перевода, выявления закономерностей и т.д. Мы продемонстрируем несколько специализированных методов автоматической обработки текста, которые в разных формах уже очень широко применяются в различных текстовых редакторах и текстовых процессорах.

Регулярные выражения.

Одним из наиболее эффективных средств автоматической обработки текста являются *регулярные выражения*.

По сути, регулярное выражение – это шаблон, который можно сравнить с фрагментом текста. Регулярные выражения могут использоваться для поиска в тексте, выборки фрагментов и их преобразования. Таким способом можно найти и выделить:

1. Имена собственные – фамилии, имена, отчества людей, названия городов, рек и т.д.
2. Численные факты – объемы, количество чего-либо и т.д.
3. Данные, соответствующие шаблону – телефоны, адреса и т.п.
4. Фразы, построенные определенным образом.

Регулярные выражения в разных видах реализованы в составе:

- текстовых редакторов и процессоров. Частичный механизм регулярных выражений реализован даже в Microsoft Word, более функциональный механизм есть в других текстовых редакторах, например в Notepad++ или в PsPad.

- специализированных средства работы с регулярными выражениями. Как одно из удобных и свободных средств такого рода можно назвать Expresso, основанный на использовании механизма регулярных выражений среды .Net. Средства такого рода позволяют отслеживать исполнение выражения, результаты его применения к тексту, создавать библиотеки готовых шаблонов.

- в составе программных библиотек, реализующих регулярные выражения в средах разработки, языках сценариев и др. Фактически, подавляюще большинство современных систем либо имеют такие средства в своем составе (PERL, PHP, JavaScript), либо они могут быть без затруднений добавлены.

В специальных аналитических системах, предназначенных для автоматической обработки текста, кроме регулярных выражений используются специальные словари и заранее подготовленные описания логических конструкций. В нашем распоряжении таких инструментов нет, но ничто не мешает выполнить анализ самостоятельно, сконструировав нужные выражения.

Регулярное выражение составляется как описание последовательности символов. Символы, которые не относятся к специальным, считаются частью шаблона. В простейшем случае это просто строка, которую мы ищем. Например:

«Иван Грозный»

Такое выражение, что очевидно, позволит найти в тексте все вхождения подстроки «Иван Грозный».

Помимо обычных символов, можно обозначить некоторые специальные. Для их записи перед ними ставится знак «\»:

Обозначение	**Символ**
\t	Табуляция
\r	Возврат каретки
\n	Новая строка
\a	Звонок
\\	Слэш
\.	Точка
^	Начало строки
$	Конец строки
\b	Граница. Первый или последний символ слова

Также действует правило экранирования: если нужно использовать символ, имеющий специальное значение, то перед ним ставится \.

Но постановка нашей задачи анализа текста такова, что мы, очевидно, не сможем задать все возможные подстроки для поиска имен. Кроме того, такая подстрока не подходит даже в том случае, когда между «Иван» и «Грозный» будет разрыв строки или два пробела.

Начнем с простого случая – обработки более сложного разрыва. Для этого мы опишем символы, которые могут стоять между словами. Для описания такого набора в регулярных выражениях предусмотрены *символьные классы*. Класс описывается набором символов в квадратных скобках. При записи можно либо перечислить подряд все символы класса, либо указать диапазон. Если сразу после открывающей скобки стоит символ ^, то класс состоит из всех символов НЕ перечисленных в скобках. В остальных местах этот символ означает начало строки. Символ точка в классе обозначает именно точку – а не любой символ. Некоторые классы уже описаны и можно использовать их сразу:

Класс[1]	**Обозначение**	**Описание**
`[A-Za-zA-Яа-я0-9_]`	[:alnum:] или \w для PCRE[2]	Символы, образующие слово
`[^\w]`	\W для PCRE	не алфавитно-цифровой или '_' символ
`[ \t\v\r\n\f]`	[:space:] или \s для PCRE	Пробельные символы
`[^\s]`	\S для PCRE	Не-пробельные символы
`[a-zа-я]`	`\p{Ll}, [:lower:]`	Прописные буквы
`[A-ZA-Я]`	`\p{Lu}, [:upper:]`	Заглавные буквы
[0-9]	`\p{N}, [:digit:],` \d – для PCRE	Цифры
[^\d]	\D для PCRE	Все не-цифры
	`. (символ «точка»)`	Любой символ, кроме символа новой строки

[1] Классы описаны для русского и английского языков.

[2] Perl Compatible Regular Expression - регулярные выражения, совместимые с языком Perl. Один из наиболее стандартизированных и развитых диалектов регулярных выражений.

Конструкции вида \p{} относятся к использованию Unicode-кодировки. Они позволяют выделить символы с определенными свойствами во всех алфавитах. Для работы с русскими буквами предпочтителен этот способ, поскольку большинство реализаций регулярных выражений не включает русские буквы в обычные классы по соображениям совместимости.

Пример:

«Иван[\s]Грозный»

Такая конструкция подойдет для тех случаев, когда слова разделяются табуляцией или разрывом строки, но символ-разделитель будет только один.

Для решения этой проблемы используются *квантификаторы* – указания на количество вхождений, которые обозначаются следующими символами:

Символ	**Значение**
*	Любое количество, включая 0
?	Может быть (не более одного вхождения)
+	Не менее одного вхождения подстроки
{a}	Ровно *a* вхождений
{a,b}	От *a* до *b* вхождений

В нашем случае конструкция будет такой:

Иван[\s]+Грозный

Но основная проблема не решена: мы не можем перечислить все имена. Остается перечислить признаки имени собственного и, исходя из них, сформировать регулярное выражение.

Имя собственное, как мы знаем, начинается с большой буквы и (в русском языке) не содержит цифр. Поэтому мы можем использовать для выделения имени собственного такую конструкцию:

\p{Lu}+\p{Ll}+

Т.е. обязательно в начале серии букв как минимум одна большая буква (сокращение в скобках от Letter Upper case), а потом обязательно хотя-бы одна маленькая (сокращение в скобках от Letter Lower case). Первая же проверка покажет, что этот шаблон выделит и все слова в начале предложений. Кроме того, при использовании этой конструкции мы захватим и один лишний символ – разделитель.

Придется их исключить. Для этого мы можем использовать при выделении нужной нам части выражения группировку и предварительные условия – так называемые *высказывания*. Особенность условий в том, что они не становятся частью захваченной части текста. Группировка выполняется с помощью скобок, а чтобы указать что группа – это предварительное условие и захватывать его не нужно используют обозначения:

Обозначение	**Значение**
?<!	Не содержит предварительно выражения…
?<	Содержит предварительно выражение…
?=	После содержит…
?!	После не содержит…

В итоге получаем:

(?<![\.]\s+)(\p{Lu}+\p{Ll})

Конечно, этот способ небезупречен. Например, нельзя распознать идущие подряд имя и фамилию. Регулярные выражения позволяют нам рассматривать разные варианты, записываемые через символ |. Обратите внимание, между словами с большой буквы должен быть либо пробел, либо тире:

(?<![\.]\s+)((\p{Lu}+\p{Ll}+[\s\-]+\p{Lu}+\p{Ll}+)|(\p{Lu}+\p{Ll}+))

Анализ достаточно длинного и разнообразного текста покажет, что и этот вариант нуждается в улучшении. Во-первых, предложение может начинаться не только после точки, но и находиться внутри текста как цитата – в кавычках. Во-вторых, имена собственные бывают и более чем из двух частей. Перечислять варианты было бы затруднительно, поэтому мы снова воспользуемся группировкой и укажем, что группы повторяются:

(?<![\.^$\"]\s+)((\p{Lu}+\p{Ll}+[\s\-]{1})+)

Существенным плюсом регулярных выражений является то, что для их работы не требуется словарь, то есть мы выявим в тексте даже те факты (например, имена), о которых ничего не знали раньше. Конечно, для этого запись текста должна, по возможности, соответствовать правилам набора, например, правилам машинописного набора, правилам записи телефонов и адресов и т.п.

Частотный анализ текста.

Регулярные выражения – мощный многоцелевой механизм обработки символьных строк. Область их применения очень велика, но для целого ряда задач они могут быть только вспомогательным средством.

Регулярные выражения сами по себе не могут применяться для анализа структуры текста, связи его с другими текстами, вынесения каких-либо суждений о тексте в целом. Для этого требуются более сложные методы анализа.

Под словом *анализ* мы будем понимать действия, направленные на выявление каких-либо существенных особенностей исследуемого объекта. При этом мы будем придерживаться общего плана:

1. Задание цели анализа, т.е. зачем мы выявляем особенности и параметры. Анализ никогда не выполняется «просто так». Больше того, результат анализа *всегда* зависит от целей.
2. Определения. Перед началом процедуры мы должны четко определить объект исследования, способ его представления, признаки, которые будем выявлять. Определить все это нам нужно в терминах предметной области – примерно так, как мы определяли понятия «слово» и «предложение» в терминах текста, т.е. набора букв.
3. Выбор метода анализа. Метод анализа подбирают исходя из целей и выделенных ресурсов (например, времени). Метод, как правило, имеет ограниченное применение, поэтому важно точно определить – какие методы мы можем использовать.
4. Планирование процедуры. Планирование включает в себя строгое описание наших действий, включая критерии достижения результата (или его недостижимости).
5. Подготовка материалов. Исходный объект редко готов к обработке сразу. Как правило, необходима предварительная подготовка – сбор, очистка, преобразование данных к виду, предусмотренному методом.
6. Собственно процедура анализа. Процедура должна завершиться получением годных для интерпретации результатов, например, числовых значений параметров.
7. Формулирование выводов. Во время этого этапа нужно на основе полученных результатов сформулировать выводы, соответствующие целям анализа.

Один из самых простых и действенных методов анализа текстов – частотный анализ. Этот метод подразумевает подсчет количества (или относительной частоты) упоминаний заданных понятий в тексте и выводы на основе полученных данных.

С помощью этого метода можно сделать выводы о направленности текста, сделанных в нем акцентах и т.п. Разумеется, не все тексты пригодны для такого анализа. Рассмотрим пример:

1. Цель – проанализировать серию художественных произведений для выявления главных и второстепенных персонажей.
2. Объектом анализа будет набор текстов в виде текстовых файлов, соответствующих правилам машинописной записи и приведенным выше определениям слов, предложений, знаков препинания. Анализ будет проводиться на основе имен собственных – имен людей.
3. Метод анализа – частотный анализ имен собственных. С достаточно высокой степенью уверенности можно сказать, что чем чаще герой будет упоминаться в тексте, тем большую роль в повествовании он играет. В простейшем случае мы можем ограничиться подсчетом прямых упоминаний (хотя можно сформулировать ряд правил для выделения и косвенных упоминаний).
4. Подготовка материалов – обработка текстов по выше описанным правилам.
5. Процедура анализа будет выглядеть так – мы выделим имена собственные, уберем всё, что не относится к именам людей, учтем склонения имен (приведем все к единой форме) и сгруппируем их с помощью любого средства.
6. Выводы мы можем сделать на основе как общего количества упоминаний, так и упоминания героев в смысловых отрывках (например, в главах). Главные герои будут упоминаться по всему тексту, эпизодические – в некоторых отрывках.

Пример:

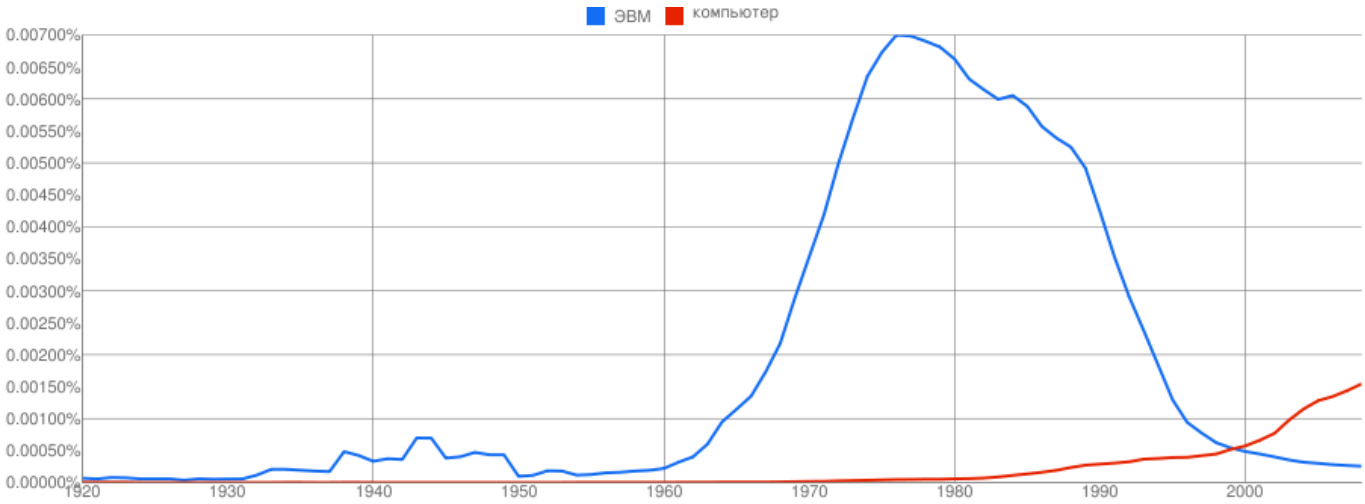


Рис. 6.7.

На рис. 6.7. приведены графики относительной упоминаемости[3] слов «ЭВМ» и «компьютер» в русскоязычных книгах за период с 1920 по 2010 год. Из графиков можно уверенно предположить, что сокращение «ЭВМ» упоминается все реже и постепенно меняется на слово «компьютер».

[3] Данные сервиса http://ngrams.googlelabs.com/ – результат анализа книг в хранилище Google Books.

Конечно, анализ художественного текста таким способом вряд ли будет представлять серьезную практическую ценность. Но вот проведенный таким способом анализ группы новостных сообщений уже может быть практически полезен, например, для того чтобы определить какие факторы (или субъекты событий) влияют на их формулировки.

Частотный анализ, помимо прочего, позволяет нам судить – относится ли текст к какой-то заранее заданной группе текстов[4]. При этом опираются на эмпирически выведенные закономерности, получившие название *закономерности Зипфа*[5].

Первый закон Зипфа:

Произведение вероятности обнаружения слова в тексте на его ранг частоты (ранг частоты наиболее частого слова равен 1) – константа[6].

Второй закон Зипфа:

Форма кривой зависимости частоты и количества слов, входящих в текст с этой частотой, одинакова для всех текстов.

То есть практически на любом достаточно большом объеме текста слово с рангом **2** встречается приблизительно в два раза реже, чем слово с рангом **1**. Слово с рангом **3** – в три раза реже слова с рангом 1 и т.д. При обработке группы текстов чаще всего подсчитывают значение частоты на миллион словоформ.

Эти закономерности, подмеченные изначально без всякой автоматической обработки, позволили обосновать следующие правила:

1. Наиболее часто встречающиеся в текстах слова не несут характеристической, смысловой нагрузки.
2. Для достаточно устойчивых типов текстов можно вывести частотные характеристики (фактически – относительный порядок) специфических слов. Эти слова будут находиться примерно «в середине» частотной кривой (часто встречающиеся слова – предлоги и т.п., редко встречающиеся – уникальные слова).

Эти эвристические правила активно используются в поисковых системах, системах классификации и других системах автоматической обработки текстов.

На основе частотного анализа построено несколько мощных методов анализа текстов на естественном языке.

Контент-анализ. Суть этого метода состоит в оценке общего направления действий (или вычленения текстов, соответствующих данному направлению) на основе придания объектам (людям, местам, событиям) определенной «окраски» в тексте. «Окраску», как правило, определяют на основе прилагательных, устойчиво связываемых с этими элементами (точнее – со словами, их обозначающими). Ручной контент-анализ – трудоемкая задача, чреватая большим количеством ошибок. Автоматические средства выполнения такого анализа должны не только вычленять отдельные слова, но и определять структуру предложений.

Существенным этапом при выполнении такого анализа является формирование специализированного словаря-тезауруса, с помощью которого можно оценить «окраску» того или иного упоминания. Такая оценка зависит не только от простого упоминания слова, но и от целевой аудитории текста, времени его написания, способа употребления. Составление такого словаря – достаточно субъективная задача, но законы Зипфа упрощают ее решение.

---

[4] Характеристический набор текстов для языка или вида текстов называется *корпусом текстов*.

[5] Выведены и предложены профессором-лингвистом Гарвардского университета Джорджем Кингсли Зипфом (George Kingsley Zipf) в 1949 году.

[6] Для русского языка – примерно 0,06-0,07, для английского – примерно 0,1.

Контент-анализ применяется и в более упрощенном виде, например для определения общего содержания сообщения, для оценки изменчивости в группе сообщений и т.д. Особую ценность этот метод имеет в связи с тем, что это один из немногих способов применить точные, количественные методы анализа и доказательства в традиционно гуманитарных областях знания.

Практикум

В практической части мы решим несколько задач, которые основаны на автоматической обработке текста. Поскольку задачи обработки текста очень разнообразны, мы не можем, конечно, утверждать, что показанный нами инструментарий – самый лучший и единственно верный. Тем не менее, у нас есть основания утверждать, что он достаточно мощен, чтобы решать с его помощью массу полезных задач.

Основной инструментарий

Основным средством, которое мы будем использовать для обработки текста, будет реализация языка Pascal – PascalABC.Net. Напомним, что получить его можно по адресу: http://pascalabc.net/ssyilki-dlya-skachivaniya.html

Помимо рабочей среды языка, нам потребуется программа подготовки и тестирования регулярных выражений Expresso. Программа эта бесплатна, и ее можно скачать по адресу http://www.ultrapico.com/ExpressoDownload.htm

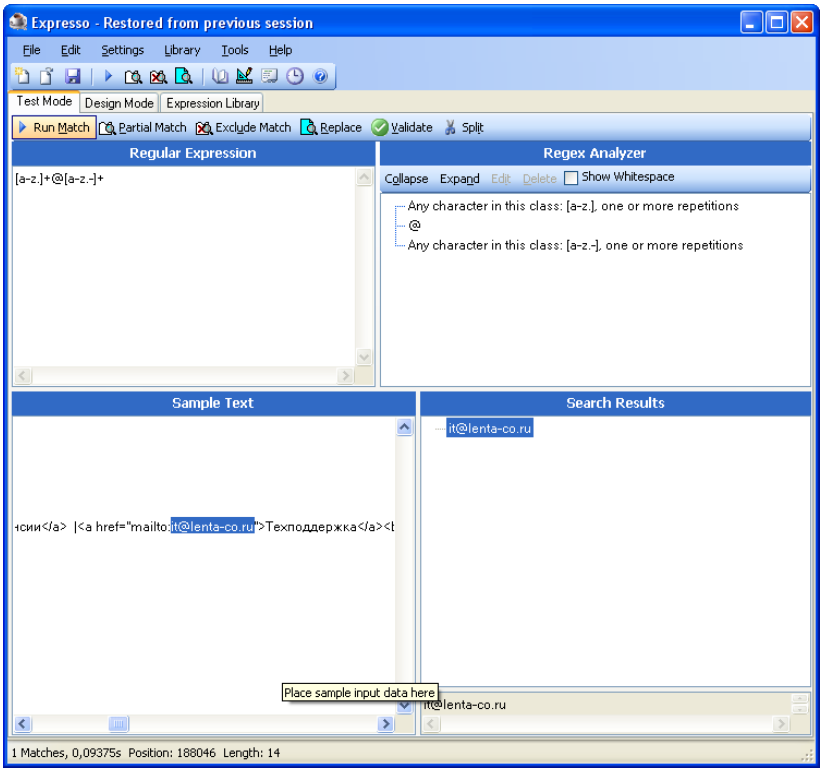
В этом разделе мы будем опираться на возможности уже не столько самой системы, сколько на возможности дополнительных библиотек – как стандартных для платформы .Net, так и дополнительно полученных.

Поскольку авторы ограничены "газетным" вариантом - некоторые вопросы (такие, как например использование лингвистических средств) в этот вариант практикума не войдут.

Выделение последовательностей по шаблону

Задача найти в тексте часть, соответствующую шаблону (или проверить соответствует ли строка шаблону), возникает очень часто и в самых разных ситуациях. Основным способом ее решения, как мы уже говорили в учебнике, является механизм регулярных выражений.

Основным средством, которое мы будем использовать для создания и проверки регулярных выражений будет программа Expresso. Начнем с краткого разбора ее интерфейса:



Основное рабочее окно программы – набор из трех страниц: Test Mode (режим проверки), Design Mode (режим создания) и Expression Library (библиотека выражений). В основном мы будем работать на первой – в режиме тестирования.
На этой странице есть во-первых управляющая панель, а во-вторых несколько разделов:

- Regular Expression – само регулярное выражение. В этом месте мы можем написать выражение – просто как текст.
- Regex Analyzer – анализатор выражения. В этом окне выражение будет разобрано на части так, как его будет интерпретировать механизм. Там можно проверить синтаксис. Если выражение сложное, то будет показано дерево – чтобы можно было проверить вложение частей.
- Sample Text – пример текста. В этот раздел загружается текст, к которому и применяется регулярное выражение. Текст можно просто скопировать туда.
- Search Result – раздел, в котором будет показан результат применения – т.е. поиска – регулярного выражения. Там будут показаны все найденные вхождения, и если вхождение выбрать в примере текста будет показано где именно оно найдено. Если выражение включает в себя несколько групп, то результат можно развернуть – опять таки, как дерево.

Решим несколько задач, опираясь на сведения о синтаксисе регулярных выражений. Во всех задачах нам нужно будет написать выражение, проверить его, выполнить и оценить результат.

Задача 1.

Выбрать из текста все целые числа
Начнем с того, что выясним – из чего может состоять целое число? Очевидно, в основном - из цифр. Кроме этого, прямо перед числом может быть знак – или + (а может их и не быть. Кроме этого, последовательность цифр может стоять в начале или в конце отдельного слова – но такая конструкция (например, «Иванов-2й») числом не будет.

Используем для этого выражения символьные классы и символы повторения:

```
[+-]?\d+
```

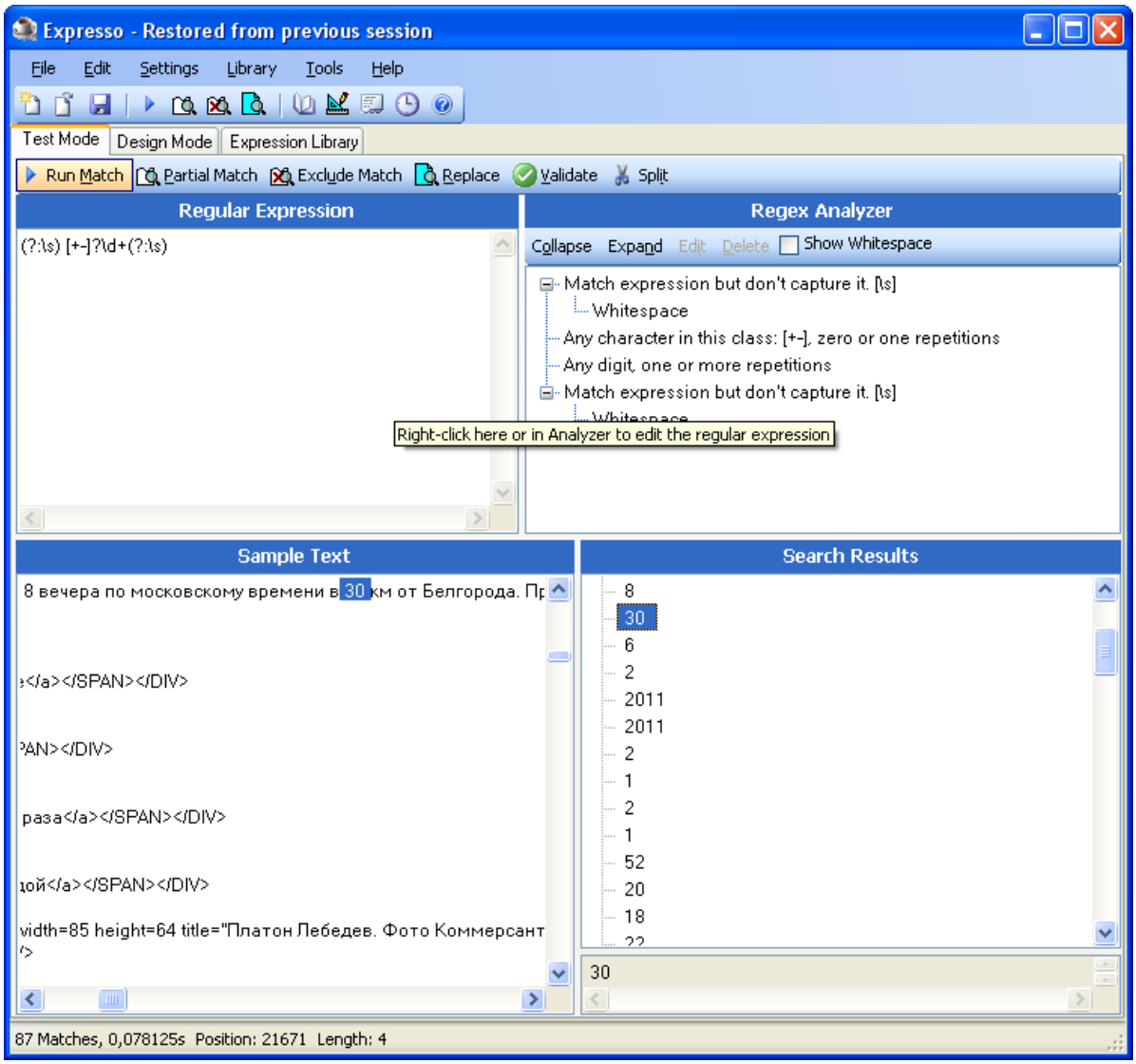
Выражение очень простое: первый его символ это либо +, либо -, это мы и записали в его классе. Он может возникнуть один раз, а может не возникнуть – поэтому выбран квантификатор ?, то есть необязательный элемент.
Следом идет набор цифр (символьный класс \d), причем в нем минимум одна цифра.

Применив это выражение мы выясним, что оно действительно находит все правильно написанные числа, но ситуацию «Иванов-2й» обрабатывает неверно – выделяет из этой подстроки число 2.

Чтобы исключить эти ситуации мы воспользуемся т.н. *незахватывающей проверкой*: то есть конструкцией, которая проверит часть строки на соответствие шаблону, но не выделит эту часть строки из результата. Синтаксис этой конструкции таков: (?:<выражение>) – мы проверяем, что <выражение> есть, но не включаем его в результат. Получится примерно вот что:

```
(?:\s)[+-]?\d+(?:\s)
```

То есть мы потребовали, чтобы перед и после нашего числа стояли пробельные элементы (класс \s).
Вот что мы получили при тестировании, просто записав HTML-код страницы в виде текста:



Обратите внимание: в разделе Regex Analyzer подробно разобрано в каком порядке и как анализируется текст в этом регулярном выражении.
Внимательно посмотрев что именно выделено, мы поймем что нашли не только нужное, но и пробелы вокруг. Чтобы все-таки выделить только целое число, придется заключить нужную часть в круглые скобки – тогда первая группа внутри найденного совпадения как раз и будет нужным нам числом:

(?:\s)([+-]?\d+)(?:\s)

Теперь рядом с каждым совпадением появится знак развертывания ветки – в которой и будет нужная группа.

Задания для самостоятельной работы:

1. Напишите выражение, которое найдет числа в составе слов.
2. Напишите выражение для поиска чисел в формате числа с плавающей точкой
3. Напишите выражение для поиска телефонных номеров.

Задача 2.

Выбрать из текста все упоминания весов в метрической системе (например, 5 килограмм, 10 тонн и т.д.). Для упрощения задачи считать, что все веса – целые, а меры написаны без сокращений.

Определим структуру текста, который нам нужно найти: он состоит из числа и меры, которая за ним записана. Как искать число – мы разобрали в прошлой задаче.

В метрической системе мер меры образуются из основного слова и десятичной приставки. Все десятичные приставки мы использовать не будем, но некоторые перечислим:

ратное	Название	Кратное (СИ)	Название	Кратное	Название	Кратное (СИ)	Название
10^0	**тонна**	10^6	мегаграмм	10^0	**тонна**	10^6	мегаграмм
10^1		10^7	*(нет)*	10^{-1}	децитонна (центнер)	10^5	*(нет)*
10^2		10^8	*(нет)*	10^{-2}	сентитонна	10^4	*(нет)*
10^3	килотонна	10^9	гигаграмм	10^{-3}	миллитонна	10^3	килограмм
10^6	мегатонна	10^{12}	тераграмм	10^{-6}	микротонна	10^0	грамм
						10^{-3}	миллиграмм

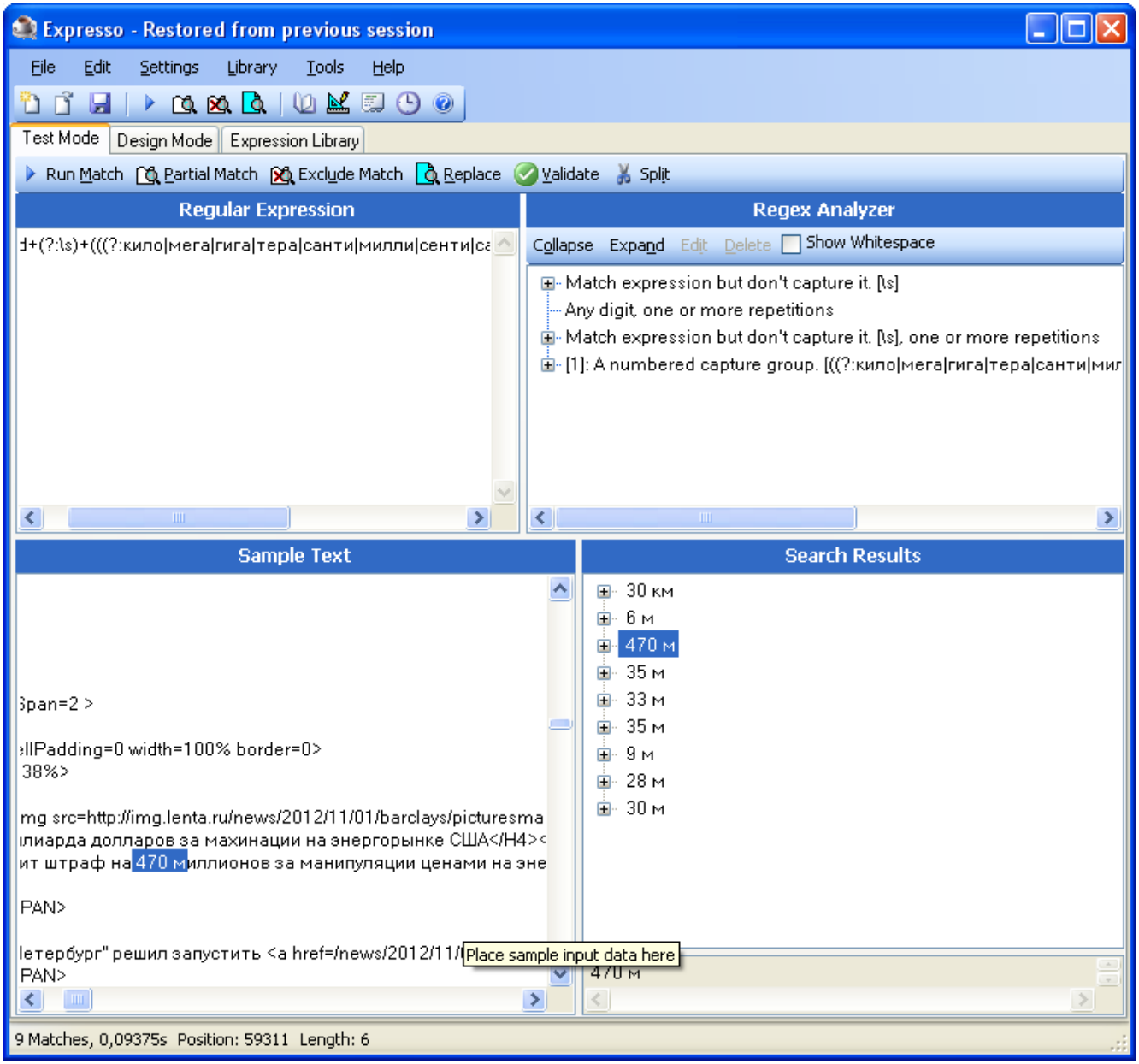
Чтобы реализовать эту структуру мы используем выбор из нескольких вариантов:

(?:\s)\d+)(?:\s)+(((?:кило|мега|гига|тера|санти|милли|сенти|санти|микро)?(?:грамм|тонн))|(г|см|м|км))

Напомним, что знак вопроса после группы задает возможное (но не обязательное) появление.

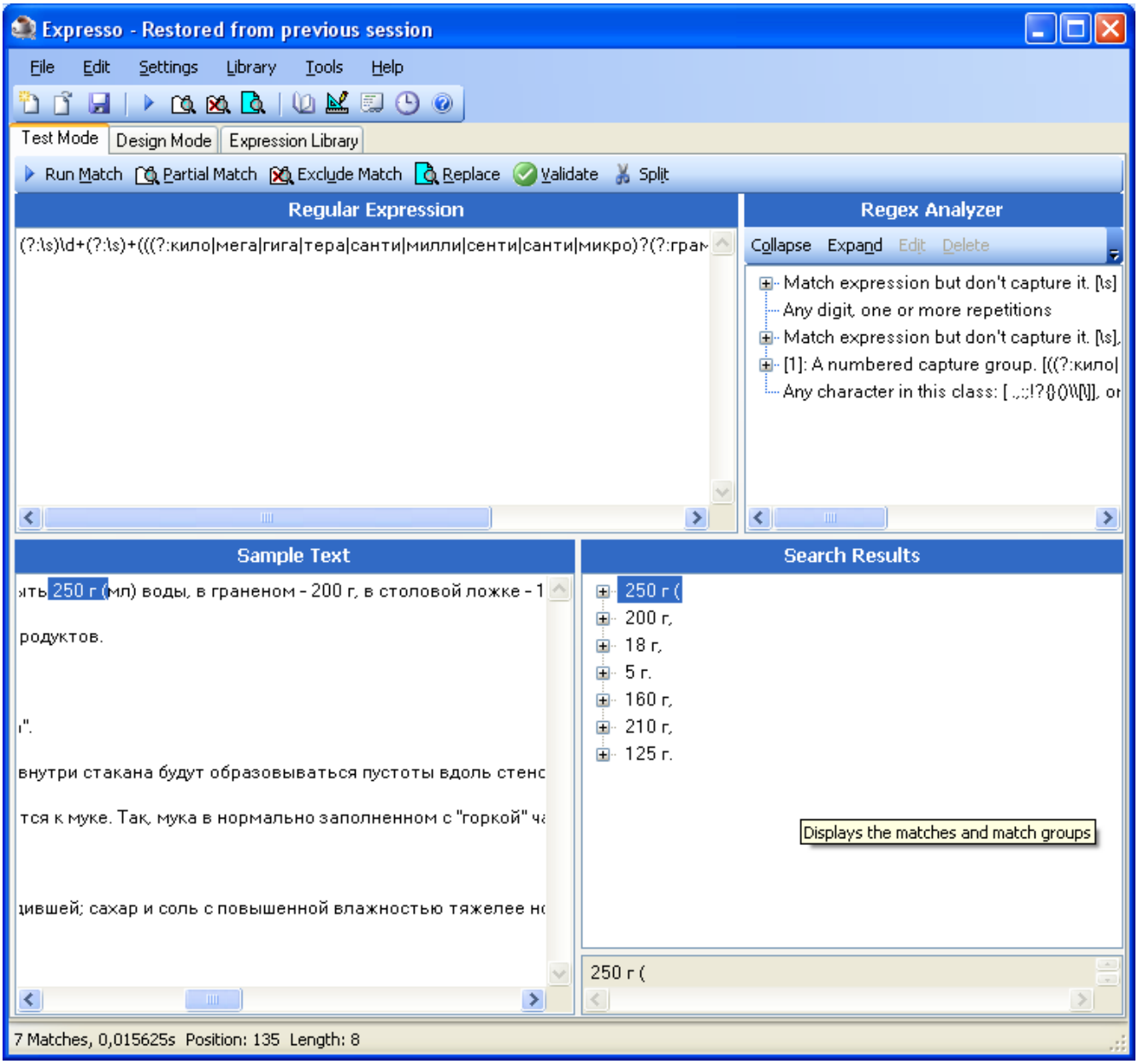
Как видите, кроме десятичных приставок мы добавили сокращения – на них правила записи не распространяются, поэтому придется перечислить их.

Применив это выражение мы выясняем, что оно действительно выбирает необходимое, но не только:



470 м – записано как мера веса, но речь в тексте – о размере штрафа. Поэтому нам придется добавить дополнительное условие в конец: расширенное множество символов:

(?:\s)\d+(?:\s)+(((?:кило|мега|гига|тера|санти|милли|сенти|санти|микро)?(?:грамм|тонн))|(г|см|м|км))[.,:;!?{}()\\[\]]+



Задания для самостоятельной работы:

1. Внесите изменения так, чтобы из найденного можно было выделить ТОЛЬКО количество и меру веса.
2. Напишите аналогичное выражение для выбора мер расстояния

Задача 3.

Выбрать из текста все адреса электронной почты

Как обычно, первое, что мы делаем, это выясняем: из каких символов состоит адрес? Согласно стандарта, адрес может состоять из двух частей: имени пользователя и имени почтового отделения. Имена разделяются в адресе знаком @. Так что этот знак – будет обязательно. Имена и слева и справа могут состоять из символов латинского алфавита, цифр, точки, знака тире и подчеркивания. Регистр букв имеет значение.

Стандарт допускает использование и ряда других символов, но на нашу работу это повлияет мало.

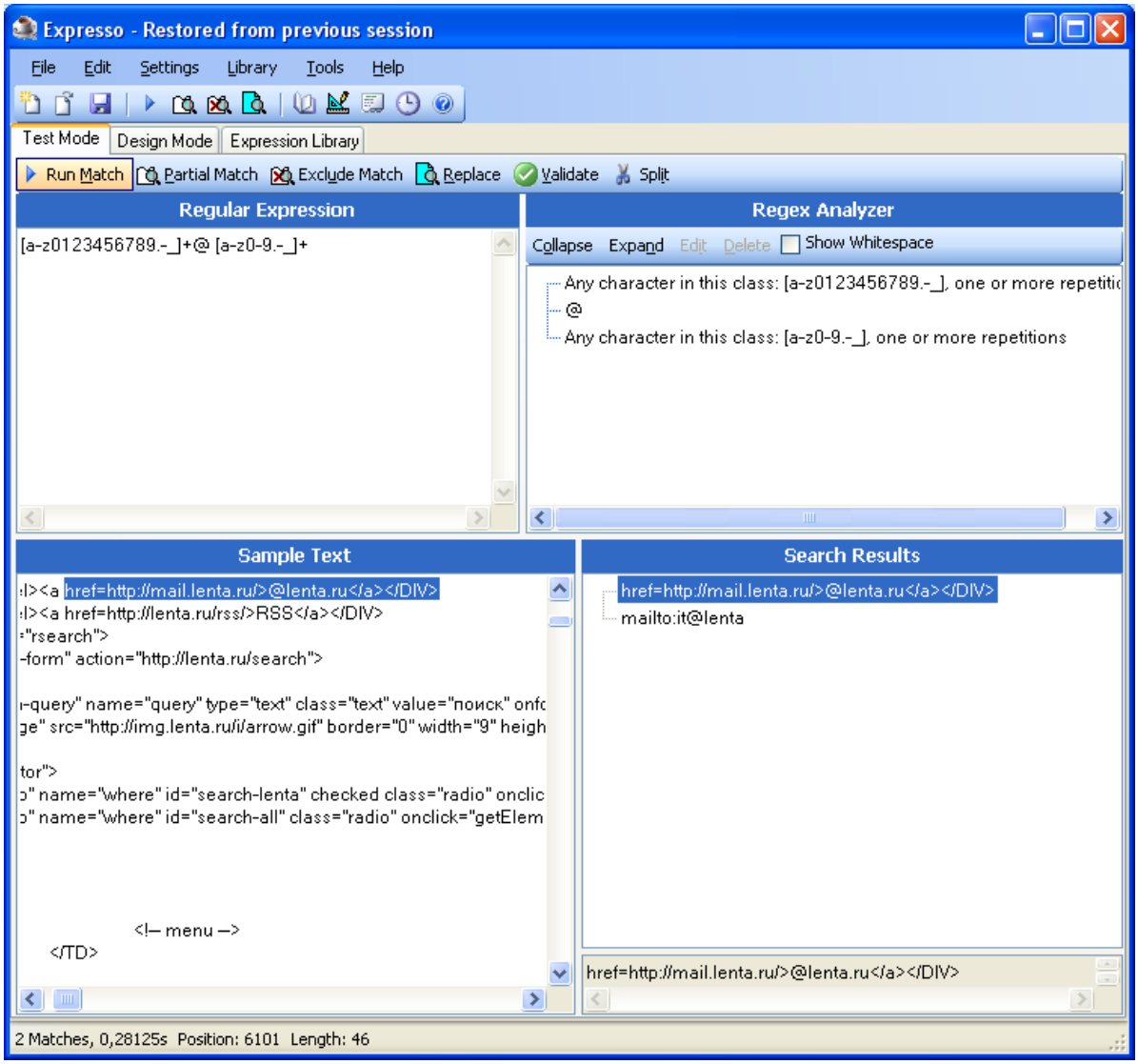
Поэтому наше выражение будет состоять из трех частей:

1. Множества символов – названия ящика. Причем название непустое – минимум один символ из этого множества должен быть
2. Знака @
3. Множества символов – названия почтового отделения

Множество символов запишем так: [A-Za-z0-9.-_]. Поскольку оно встречается минимум один раз, то зададим повтор знаком +
В итоге, выйдет примерно следующее:

[A-Za-z0-9.-_]+@[A-Za-z0-9.-_]+

Для проверки скопируем реальный текст в окно примера и применим выражение. Текст мы взяли из html-кода страницы сервера новостей, и применив увидели примерно вот что:



Как видим, адреса выделены неправильно. Причина – появились лишние символы-разделители. Откуда они взялись? Проанализируем запись множества и обнаружим, что добавили лишний диапазон: .-_. Мы имели в виду просто символ -, но он имеет в записи множеств специальный смысл. Поэтому перед его упоминанием ставим знак \.
Теперь результат будет верен, он станет примерно таким, как на первой иллюстрации. Теперь выражение будет выглядеть так:

[A-Za-z0123456789.\-_]+@[A-Za-z0-9.\-_]+

Тем не менее, этого недостаточно. Название почтового ящика не может начинаться с точки и содержать две точки подряд. После знака-разделителя тоже не может стоять точка.
Решим эту задачу так: разобьем последовательность на группы:

[A-Za-z0123456789\-_]+(?:\.[A-Za-z0123456789\-_])*@[A-Za-z0123456789_]+(?:\.[A-Za-z0123456789\-_])+

Обратите внимание: в обоих случаях, в начале идет группа без точки, а потом некоторое количество групп, которые начинаются с точки. Если в первой части – таких групп может и не быть, то во второй – они есть обязательно.

Задания для самостоятельной работы:

1. Напишите регулярное выражение для выборки из текста дат в формате ДД/ММ/ГГ или ДД/ММ/ГГГГ. В выражении учтите, что в датах не бывает номера дня больше 31 и месяцев – всего 12.
2. Вы получили текстовый файл протокола обращений пользователей к ресурсам среды Интернет, в котором построчно перечислены атрибуты каждого обращения через запятую. Порядок полей совпадает, названия полей заданы в первой строке. Подготовьте регулярное выражение, позволяющее отфильтровать только 1, 4 и 7 столбцы и перечисляющее значения через точку с запятой.
3. Выделите с помощью регулярного выражения все имена собственные из географического текста (например, реферата).

Использование регулярных выражений при подготовке программ

Теперь рассмотрим, каким образом мы сможем использовать механизм регулярных выражений в своих программах. Стандарт Паскаля не предусматривает библиотек для работы с ними, но в учебной среде нам доступна библиотека, входящая в комплект платформы .Net. Подключим ее в модуле Uses, и получим возможность использовать объект Regex и необходимый тип Match – совпадение.

```
program RegularExperssionsMatch;
uses System.Text.RegularExpressions;

var
  s: String;
  txt: Text;
  m: Match;

begin
  Assign(txt, 'k:\demo.txt');
  Reset(txt);

  while not Eof(txt) do
  begin
    ReadLn(txt, s);
     foreach m in Regex.Matches(s, '[A-Za-z0123456789\-_]+(?:\.[A-Za-z0123456789\-
           _])*@[A-Za-z0123456789_]+(?:\.[A-Za-z0123456789\-_])+') do
      Writeln('Найдено:', m.Value);
  end;
  Close(txt);
end.
```

Обратите внимание на конструкцию ***foreach… in…*** – с ее помощью мы получаем возможность перебрать все элементы во множестве-коллекции. Эта синтаксическая конструкция отсутствует в классическом языке Паскаль, но есть в расширенном (Extended Pascal, ISO 10206:1990).

Задания для самостоятельной работы:

1. Напишите программу поиска заданного слова в текстовом файле. Считайте, что текст записывается без использования переносов
2. Напишите программу для поиска двух идущих подряд заданных слов. Учтите, что между словами может быть несколько разделителей – например, два и более пробелов.
3. Напишите программу, подсчитывающую количество упоминаний в текстовом файле семьи Ивановых (или любого из её членов). Считайте, что фамилия во всей семье одна.

Помимо поиска совпадений, регулярное выражение может использовать для разделения строки на части. В этом случае регулярное выражение будет использовать для поиска последовательностей-разделителей.
В этом примере с помощью этой возможности строки делятся на слова – то есть последовательности символов между последовательностями разделителей.

```
Program RegularExperssionsSplit;
uses System.Text.RegularExpressions;

var
 s,s0 : String;
 txt: Text;

begin
  Assign(txt, 'k:\demo.txt');
  Reset(txt);

  while not Eof(txt) do
  begin
    ReadLn(txt, s);
    foreach s0 in Regex.Split(s, '[ ,.!?{}()\[\]/*+\-]+') do
      Writeln(s0);
  end;

  Close(txt);

end.
```

Задания для самостоятельной работы:

1. Напишите программу подсчета общего количества слова в текстовом файле. Считайте, что текст записывается без использования переносов, числа не считаются словами.
2. Напишите программу обработки текстового файла с переносами, которая уберет переносы – то есть восстановит разбитые слова и результат сохранит в файл, добавив к имени .new. Учтите, что знак, обозначающий перенос, может применяться и в других случаях.

Еще один метод, который нам может понадобиться – замены. Этому методу нужно будет передать второе выражение, которое определит что должно оказаться на месте найденного фрагмента.
Приведем пример, в котором по всему текстовому файлу заменим квадратные скобки на угловые.

```
Program RegularExperssionsReplace;
uses System.Text.RegularExpressions;

var
 s,s0 : String;
 txt: Text;

begin
  Assign(txt, 'k:\demo.txt');
  Reset(txt);

  while not Eof(txt) do
  begin
    ReadLn(txt, s);
    s0 := Regex.Replace(s, '\[.+\]', '<$0>')
    Writeln(s0);
  end;

  Close(txt);

end.
```

Задания для самостоятельной работы:

1. Вы получили текст, в котором все даты записаны в виде ГГГГ-ММ-ДД (например: 2010-12-31 – 31 декабря 2010 года), или ДД-ММ-ГГ, или ДД-ММ-ГГГГ. Напишите программу которая, преобразует все эти даты к привычному нам виду: ДД.ММ.ГГГГ (т.е. 31.12.2010), и результат сохранит в новый текстовый файл.
2. Вы получили текст, в котором записаны номера телефонов в разных формах. Примеры из текста: +7 (495) 123-4567, 123-4567, 1234567, 499-1234567. Напишите программу которая, преобразует все эти варианты к общему стандартному: +7 (495) 123-4567, и результат сохранит в новый текстовый файл. По умолчанию (если не указано) предполагаем, что номера российские, код города – на Ваше усмотрение.

Задача 1. Частотный анализ

Идея частотного анализа очень проста: если у нас есть набор каких-то элементов (букв, слов, предложений, яблок – чего угодно), то простой способ оценить этот набор – посчитать сколько в этом наборе элементов каждого вида (каждой буквы, слова, размера яблок). Узнав какую долю (то есть насколько часто) встречается элемент мы можем делать какие-то выводы. Первая группа задач, которую мы будем решать, будет связана с кодировками и алфавитом.

Дан текст на русском языке, в кодировке Windows 1251, длина заранее не известна, в форме файла. Требуется: составить частотную таблицу упоминаемости символов в этом тексте.

Решение:

Нетрудно увидеть, что основной частью этой задачи будет посчитать количество упоминаний каждого символа в тексте. Вспомним определение таблицы кодировки и особенности конкретной кодировки Windows-1251. Нам точно известно, что в ней всего 256 символов, с номерами от 0 до 255 включительно.

Создадим массив из 256 беззнаковых (не может быть отрицательного количества) целых чисел. Будем считывать по одному символу и увеличивать значение в соответствующей (то есть с номером, равным коду символа) ячейке массива. Единственное, что нам придется сделать – преобразовать символ в его код.

Заготовка программы будет выглядеть примерно так:

```
program frequency;
```

```
var
  txt : Text;
  c : char;
  countLetters : array[0..255] of Word;
  i : Word;

begin
  Assign(txt,'demo.txt');
  Reset(txt);

  for i:=0 to 255 do
    countLetters[i] := 0;

  while not Eof(txt) do
  begin
    read(txt,c);
    inc(countLetters[ ord(c) ]);
  end;

  for i:=0 to 255 do
    WriteLn( i, ' - ',countLetters[i]);

  Close(txt);
end.
```

Найдем какой-нибудь достаточно большой текстовый файл, сохраним его под именем demo.txt и посмотрим, что получится.

Проанализировав результаты мы выясним, что:

1. Довольно много символов не относится к языку, а часть – даже не буквы (особенно, символы «возврат каретки» и «перевод строки», «пробел» и т.п.. Очевидно, они не должны влиять на частоту.
2. Если объем текста достаточно велик (около 3Мб) появляется довольно большая вероятность получить ошибку «Переполнение»
3. Поскольку нам придется считать общее количество символов, то переменная для этого точно должна допускать большие значения, чем обычный тип Word.

При подготовке программы нам нужно будет ввести фильтрацию символов. Самый простой способ, если мы работаем с русским языком – учитывать только символы расширенного набора – т.е. с кодами больше 255.

В итоге, программа будет примерно такой:

```
program frequency;

var
  txt : Text;
  c : char;
  countLetters : array[0..255] of LongWord;
  i : Word;
  sigma : LongWord;

begin
  Assign(txt,'k:\demo.txt');
  Reset(txt);

  for i:=0 to 255 do
```

```
    countLetters[i] := 0;

  while not Eof(txt) do
  begin
    read(txt,c);
    if ord(c) > 127 then
      inc(countLetters[ ord(c)]);
  end;

  sigma := 0;
  for i:=0 to 255 do
    sigma := sigma + countLetters[i];

  for i:=128 to 255 do
    WriteLn( chr(i), '(',i, ')', ' - ',(countLetters[i]/sigma):8:7 );

end.
```

Результат ее запуска для текста 1-го тома романа «Война и мир» примерно таков:

Буква	Частота	Буква	Частота
о(238)	0.1123347	В(194)	0.0020581
а(224)	0.0822470	А(192)	0.0019786
е(229)	0.0789770	П(207)	0.0019214
и(232)	0.0653834	К(202)	0.0017662
н(237)	0.0635027	О(206)	0.0015796
т(242)	0.0557268	Б(193)	0.0015168
с(241)	0.0524678	М(204)	0.0011122
л(235)	0.0501234	Д(196)	0.0010235
р(240)	0.0445219	С(209)	0.0008868
в(226)	0.0437774	Р(208)	0.0008591
к(234)	0.0346491	Т(210)	0.0008406
д(228)	0.0292490	И(200)	0.0008018
м(236)	0.0283326	Я(223)	0.0005616
у(243)	0.0283308	ъ(250)	0.0005247
п(239)	0.0236641	Г(195)	0.0004896
я(255)	0.0224928	Э(221)	0.0004360
г(227)	0.0201539	Ч(215)	0.0003991
ь(252)	0.0193780	Е(197)	0.0003399
ы(251)	0.0188958	Л(203)	0.0002697
з(231)	0.0175231	З(199)	0.0002051
б(225)	0.0156794	У(211)	0.0002014
ч(247)	0.0131742	Ж(198)	0.0001496
й(233)	0.0114783	Ш(216)	0.0001275
ж(230)	0.0099319	Ф(212)	0.0001256
ш(248)	0.0092742	Х(213)	0.0001127
х(245)	0.0083801	Ц(214)	0.0000388
ю(254)	0.0064421	Ю(222)	0.0000148
ц(246)	0.0039979	Ь(220)	0.0000074
щ(249)	0.0027897	Щ(217)	0.0000037
э(253)	0.0025735	Й(201)	0.0000018
ф(244)	0.0021024	Ъ(218)	0.0000000
Н(205)	0.0020913	Ы(219)	0.0000000

Несколько вопросов и предложений для совершенствования программы:

1. Проанализируйте вывод программы. Можно ли сделать результат ее работы более коротким, если ограничиться русским языком?
2. Можно ли сократить объем занимаемой памяти?
3. Стоит ли вместо отдельного цикла с подсчетом количества русских букв добавлять единицу к сумме сразу в основном цикле?
4. Что еще нужно учесть, чтобы получить настоящую частотную таблицу – т.е. таблицу, в которой частота подсчитана именно *для буквы*, а не *печатного символа*?
5. Предложите вариант усовершенствования, который сможет работать с кодировкой Unicode.

Очевидно, что частоты встречаемости символов в больших русских текстах хоть немного, но будут различаться. Но вот порядок символов по частоте упоминаемости будет примерно одинаковым.

Задания для самостоятельной работы:

1. Подсчитайте частоты для большого объема текстов разного вида: для официальных документов, литературных произведений. Совпадают ли они? Сохраняется ли порядок символов по частоте встречаемости?
2. Как с помощью частот встречаемости различать таблицу кодировок текста: CP866, Windows-1251, KOI-8?
3. Напишите программу преобразования текста между разными таблицами кодировки:
    a. Win1251 <-> KOI-8
    b. Win1251 <-> CP866
    c. CP866 <-> KOI-8

Задача 2. Анализ частоты упоминаемости слов в тексте

Дан файл текста на русском языке, в кодировке Windows 1251, длина заранее не известна. ,
Требуется составить частотную таблицу упоминаемости слов в этом тексте.

Решение:

При решении этой задачи нам нужно учесть несколько особенностей:

- Во-первых, мы не знаем сколько слов в тексте будет.
- Во-вторых, в отличие от символов слово нужно в тексте выделить, потому что это не единый объект, а набор объектов
- В-третьих, слово может быть написано по-разному, в зависимости от того – первое это слово в предложении или нет.
- Слово может быть разделено на части: если текст заранее отформатирован устаревшими методами.

На самом деле, трудностей больше, но решать их мы будем позже.

Выделять слова из текста мы будем с помощью регулярного выражения. Чтобы его построить, мы вспомним, что такое «слово». Нас будет интересовать строго формальное определение, которое позволит нам выделить слово из потока символов. С этой точки зрения слово – это последовательность символов, взятых из текста, и не разделителей.

Воспользуемся для создания программы во-первых методом деления строки на части по регулярному выражению, а во-вторых библиотекой работы с коллекциями из .Net для организации словаря:

```
Program Frequency;

uses System.Text.RegularExpressions,System.Collections.Generic;

var
 s,s0 : String;
 txt : Text;
 tx  : System.Collections.Generic.Dictionary<string, integer>;
 m: Match;

begin
  tx := new System.Collections.Generic.Dictionary<string, integer>;

  Assign(txt,'k:\demo.txt');
  Reset(txt);

  while not Eof(txt) do
  begin
    ReadLn(txt,s);
    foreach s0 in Regex.Split(s, '[ ,.!?(){}+*\-;:"\[\]\\]+') do
      if s0 <> '' then
            if tx.ContainsKey(s0) then
          tx[s0] := tx[s0]+1
        else
          tx.Add(s0,1);
  end;
  close(txt);

  Assign(txt,'k:\frequency.csv');
  Rewrite(txt);

  WriteLn( txt,'Word;Count;' );
  foreach s0 in tx.Keys do
    WriteLn( txt,s0, ';', tx[s0],';' );
  Close(txt);
end.
```

Запустив эту программу, мы сможем проанализировать текстовый файл и получить результаты примерно в таком виде:

Автор	2
автор	8
автора	3
авторе	1
авторитет	9
авторитета	2
авторитетами	1
авторитете	1
авторитетная	1
авторитетнее	1
авторитетной	1
авторитетную	1
авторитетом	1
авторитету	3
авторитеты	1
авторов	2

Ага	16
агитации	2
агрегата	1
агрегаты	1
агронома	1
Административная	1
административная	1
административной	1
административную	2
административные	1
административный	1
администратор	2
администратора	1
администрацией	2
администрации	1
администрировании	1

авторских	1
авторскую	1
авторство	1
автору	3
авторы	3
Авторы	1

администрировать	1

Изначально результат будет не отсортирован, но это сделать несложно.
На что нам придется обратить внимание прежде, чем мы сможем воспользоваться результатами для анализа? На несколько особенностей:

1. Учитывается регистр, а значит «Административная» и «административная» это два разных слова. Очевидно, что с точки зрения извлечения смысла - это не так.
2. Самые частые слова не несут никакого значения для анализа текста. Сложно предположить, что нам важна частота слова «Ага» или «И».
3. Мы не учитываем формы слов, поэтому, например, видим 6 вариантов одного и того же по смыслу слова.

Первая проблема решается довольно просто: нужно перед разбиением преобразовать всю строку к единому регистру – верхнему или нижнему. Метод, который это реализует – метод самой строки toLower (или toUpper – логически разницы никакой нет). Добавим в программу:

```
s := s.ToLower;
```

Куда это поместить – надеемся, читатель в состоянии определить сам.

Вторая проблема решается несколько труднее: нельзя просто выкинуть слово, потому что оно часто встречается в этом конкретном тексте. Чтобы понять какие слова не характеризуют конкретный текст нам нужно проанализировать большое количество эталонных текстов – выделить самые частые слова. К счастью, за нас это уже сделано: воспользуемся статистической информацией с сайта корпуса русского языка: раздел «Частоты», самые частые 100 словоформ будут нашим стоп-листом: http://ruscorpora.ru/1grams.top.html
Для реализации нужно будет сохранить их в файл, а в программе предусмотреть отдельную коллекцию стоп-слов, в которую загрузить слова из файла и проверять перед добавлением – есть там слово или нет.
Реализацию этого подхода авторы предоставляют читателям для самостоятельного упражнения.

Третья проблема существенно сложнее: для ее решения нам придется как минимум преобразовать слова к их исходной, базовой форме, т.е. выполнить так называемую *лемматизацию*. Для русского языка это непростая задача, которую можно решить использованием специализированной библиотеки. Как было сказано вначале в рамках сокращенного газетного варианта делать это было бы очень затруднительно - и для авторов, и для читателей.

Автоматизированная и автоматическая обработка текста, извлечение из текста данных, анализ больших массивов информации, поиск и смежные с ним задачи - современное и быстро развивающееся направление. Его достижениями мы пользуемся постоянно - когда обращаемся к любым документальным и поисковым системам, работаем со сложными сайтами. Авторы полагают, что ознакомление с базовыми приемами и средствами такой работы - необходимый элемент подготовки в области информационных технологий.