Санкт-Петербургский государственный университет

Математико-механический факультет

Информационные системы и базы данных

Коновалова Ирина Михайловна

Разработка эталонной коллекции для задачи поиска нечётких дубликатов в документации программного обеспечения

Выпускная квалификационная работа

Научный руководитель:
д.т.н., профессор Кознов Д.В.

Рецензент:
ведущий инженер-программист Луцив Д.В.

Санкт-Петербург

2018

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems Software Engineering

Information Systems and Data Bases

Irina Konovalova

Near Duplicate Benchmarks for Software Documentation

Graduation Project

Scientific supervisor:

Doctor of Engineering, Professor D.V. Koznov

Reviewer:

Lead software engineer D.V. Luciv

Saint-Petersburg

2018

Оглавление

Введение

Документация программного обеспечения (ПО) играет ключевую роль в разработке и поддержке программных продуктов. Качество документации имеет первостепенное значение для успеха проекта: взаимодействие разработчиков и пользователей происходит посредством документации, для сопровождения и развития проекта также требуется документация. Так как документация создаётся на естественном языке, то имеется недостаток средств для автоматизированной оценки её качества.

Как правило, документация доступна не только для чтения, но так же подвергается постоянным пересмотрам для адаптации к постоянно меняющимся требованиям. При этом часто происходит параллельная модификация нескольких связных частей теста, поскольку одна и та же информация представлена многократно. Таким образом, если изменения в документации не будут применены ко всем дубликатам, то могут возникнуть несогласованности, которые впоследствии затруднят понимание документации и могут привести к ошибкам и коллизиям.

Проект Duplicate Finder [1] предназначен для поиска и анализа повторов в документации ПО. Так как не все дубликаты являются дословными, то Duplicate Finder, с помощью реализованных в нем алгоритмов [2], позволяет искать как точные, так и вариативные (неточные) повторы.

Существующие алгоритмы позволяют находить повторы в документации, но необходимо понять, насколько полученный результат работы этих алгоритмов является качественным. При этом возникает задача оценки качества этих алгоритмов.

После применения алгоритмов к документации ПО пользователь получает список повторов — далее будем называть этот список данными. Существуют разные определения качества данных: одни авторы утверждают, что это релевантность (в общем смысле это соответствие потребностям пользователя, степень удовлетворения показанными в ответ на запрос

результатами) [3, 4], другие говорят, что следует оценивать точность, полноту данных и другие свойства [5, 6].

К сожалению, существующие алгоритмы находят не все дубликаты в документации ПО, а также выдают повторы, не являющиеся осмысленными. Также важным является правильное нахождение границы повтора, так как только часть повтора может содержать осмысленную семантику. Пока это тяжело сделать автоматически, поэтому для оценки качества алгоритмов поиска дубликатов целесообразно создать эталонную коллекцию, содержащую все осмысленные повторы анализируемого документа. С помощью эталонной коллекции можно проверять качество существующих алгоритмов поиска неточных повторов, запуская их на этом же документе и сравнивая результаты их работы с эталонной коллекцией. Такая проверка важна для проектирования и разработки эффективных поисковых систем, поскольку позволяет измерить насколько система отвечает своей главной цели.

Постановка задачи

Целью данной работы является создание коллекций неточных повторов для документации ПО, а также разработка инструмента для поддержки редактирования и навигации по коллекциям. Для достижения этих целей были сформулированы следующие задачи.

1. Знакомство с подходом и инструментом Duplicate Finder. Изучение правил построения и использования эталонных коллекций в задачах информационного поиска.
2. Изучение документации ПО на предмет поиска и анализа неточных повторов. Создание эталонных коллекций.
3. Проектирование и реализация системы по работе с эталонными коллекциями неточных повторов в контексте инструментария Duplicate Finder: разработка формата хранения; разработка средства редактирования; разработка средств навигации и анализа.
4. Выполнение апробации системы на реальных алгоритмах поиска неточных повторов, анализ результатов, выводы.

Глава 1. Обзор

1.1 Инструмент Duplicate Finder

В рамках данной работы используется программный инструмент Duplicate Finder [1, 2], реализованный на языке Python, который позволяет находить и графически отображать повторы в выбранном документе, а также задавать параметры поиска дубликатов. Имеется два следующих режима работы инструмента:

- автоматический — позволяет быстро определить нечёткие повторы в предложенном документе;
- интерактивный — позволяет редактировать найденные повторы; пользователь сам решает, рассматривать ли результаты работы инструмента как повтор или нет.

Duplicate Finder использует инструмент Clone Miner [7] для поиска точных повторов. На основе этих повторов Duplicate Finder реализует собственные алгоритмы. В автоматическом режиме используется алгоритм компоновки неточных повторов, в интерактивном режиме — методика интерактивного поиска неточных повторов и алгоритм поиска по образцу [2].

В результате работы инструмент отображает следующую информацию: список найденных групп повторов, их общую (архетип) и вариативную (дельта) части.

1.2 Эталонные коллекции в задачах информационного поиска

Для того, чтобы иметь возможность оценить качество данных, необходимо выбрать критерии качества — свойства данных, по которым будет приниматься решение о соответствии данных информационным потребностям. Но для каждой предметной области имеется своё понимание критериев. Так как только данные, соответствующие критериям, могут считаться качественными, то определение критериев для предметной области

является важной составляющей при оценке информационно-поисковой системы.

Одной из используемых методик оценки качества поиска является подход на основе тестовых коллекций. Впервые формальное тестирование информационно-поисковых систем было проведено в ходе экспериментах, проводимых в университете Cranfield [8] в Великобритании в 1958–1966 годах. Подход, используемый в ходе данной работы, использует тестовые коллекции — повторно используемые и стандартизированные ресурсы, которые могут использоваться для оценки поисковых систем. Эти коллекции совместно с оценочными метриками позволяют количественно сравнивать различные алгоритмы поиска и эффекты от изменения параметров алгоритма.

Создание и использование общих коллекций признано важным пунктом в сообществе информационного поиска, это подтверждают многочисленные проекты по составлению коллекций (TREC, NTCIR, CLEF, RCUI и другие). Несмотря на то, что создание коллекций требует существенных первоначальных усилий, это окупается из-за возможности её повторного использования в различных экспериментах. Тестовые коллекции помогают сравнивать различные алгоритмы, проверенные на этой же коллекции.

Вместе с созданием тестовой коллекции возникает необходимость в выборе метрики оценки качества. Чаще всего в качестве метрик используются два показателя: точность (precision) и полнота (recall). Под *точностью* понимают долю релевантных повторов среди тех, которые были найдены алгоритмом. *Полнота* — это доля найденных алгоритмом релевантных повторов среди всех релевантных.

Глава 2. Результаты анализа документации ПО и эталонные коллекции

В рамках данной работы была рассмотрена API-документация программного обеспечения, а именно, документы LibLDAP Manual и Python Requests. API-документация (Application Programming Interface) является технической документацией ПО, содержащей инструкции о том, как эффективно использовать программное API, аппаратное обеспечение (SCPI) или Web API. В ней описывается вся информация, необходимая для работы с API — классы, типы возвращаемых данных, методы и поля библиотек и т.д.

LDAP (Lightweight Directory Access Protocol) — это облегчённый протокол доступа к службам каталогов, предназначенный для доступа на основе стандарта X.500 [9]. LDAP работает поверх TCP/IP или других ориентированных на соединение сетевых протоколов. OpenLDAP — реализация LDAP с открытым исходным кодом, разработанная в рамках одноимённого открытого проекта, распространяется под собственной свободной лицензией OpenLDAP Public License. В документации LibLDAP Manual описываются основные структуры данных и стандартные функции библиотек OpenLDAP LDAP и OpenLDAP LBER. Например, функции, описанные в параграфе LDAP_MODIFY, используются для соединения с сервером LDAP и изменения записей, функции из параграфа LDAP_SEARCH — для соединения с сервером LDAP и выполнения поиска с использованием заданных параметров.

Python Requests является библиотекой для языка Python, которая позволяет отправлять HTTP-запросы без сложных «ручных» настроек. В документации представлена информация о классах и методах данной библиотеки.

Также был проведён анализ фрагмента документации Zend Framework, являющийся руководством для программистов (полная версия документации насчитывает около тысячи страниц). Но данный фрагмент оказался слишком неформальным и содержал мало повторов, описывающих программные

сущности, поэтому эти результаты оказались не интересны в рамках данной работы.

Документация анализировалась в кодировке UTF-8, то есть в формате «плоского» текста. Это представление было получено с помощью утилиты Pandoc [14] из различных исходных форматов (LibLDAP Manual был в формате TROFF, Python Requests — в формате встроенной в Python разметки, Zend Framework — в формате DocBook)[1]. Сведения о документации, выбранной для анализа, представлены в табл. 1.

№	Документ	Тип	Размер[2], Кб	Доступ
1	LibLDAP Manual	API-документация	1076	https://www.openldap.org
2	Python Requests	API-документация	36	http://python-requests.org
3	Zend Framework V1 guide	Рук-во программиста	1473	https://github.com/zendframework/zf1/tree/master/documentation

Таблица 1. Документация ПО, выбранная для анализа

Существующие автоматические алгоритмы поиска повторов не способны выделять семантику (осмысленность) найденного. Либо, даже получая осмысленный повтор, алгоритм захватывает лишние фрагменты. Осмысленность важна для переиспользования фрагментов текста, но это абстрактное понятие. При анализе документации стало понятно, что лучше всего для выделения осмысленных фрагментов подходит API-документация. При нахождении синтаксического повтора его границы было решено определять «замыкая» повтор до описания некоторого свойства ПО (software feature) — описание функции, параметра и т.д. При данном подходе теряется

[1] Подробнее об этом можно прочитать в [2].

[2] Размер указан для «плоского» текста, полученного после преобразования исходной документации утилитой Pandoc.

процентное соотношение между вариативной частью и архетипом группы, но при этом такой подход позволяет чётко идентифицировать раскопированную информацию. Основным критерием объединения повторов в группы была их семантическая схожесть.

Рассмотрим пример группы, состоящей из двух фрагментов текста и содержащей большую вариативную часть (жирным шрифтом выделены отличающиеся фрагменты текста):

minssf specifies the **minimum** acceptable security strength factor as an integer **approximating the effective key length used for encryption. 0 (zero) implies no protection, 1 implies integrity protection only, 56 allows DES or other weak ciphers, 112 allows triple DES and other strong ciphers, 128 allows RC4, Blowfish and other modern strong ciphers.** The default is **0**.

….

maxssf specifies the **maximum** acceptable security strength factor as an integer **(see minssf description).** The default is **INT_MAX**.

Это два сходных фрагмента документации LibLDAP Manual, оба являются описанием ключевых слов параметра <properties> (первый фрагмент — ключевого слова minssf, второй — maxssf) опции SASL_SECPROPS, описанной в параграфе LDAP.CONF. Оба ключевых слова задают значение коэффициента надёжности безопасности. При этом можно заметить, что у второго фрагмента имеется ссылка на первый фрагмент, из-за этого, несмотря на то, что оба фрагмента семантически похожи, дельта (вариативная часть) составляет больше 15% от длины архетипа, поэтому алгоритм компоновки Duplicate Finder не выделит данные фрагменты в группу повторов [2].

При замыкании повтора до описания свойства мы получаем не просто повторяющийся фрагмент текста, который может даже не содержать смысла, а семантически замкнутую единицу ПО, которая оказывается привязана к предметной области, а не к тексту как таковому. Эту единицу можно повторно использовать и согласованно изменять.

Общая информация о составленных эталонных коллекциях представлена в табл. 2.

Название	Среднее количество элементов в группе	Количест-во групп по 2 элемента	Количест-во точных повторов	Количество вариативных повторов	Общее кол-во найден-ных групп
LibLDAP Manual	18,1	25	1293	582	76
Python Requests	3,9	13	54	36	31

Таблица 2. Общая информация об эталонных коллекциях

В документе LibLDAP Manual большое количество точных повторов объясняется тем, что в описаниях функций и структур данных, содержатся общие разделы. Например, раздел ACKNOWLEDGEMENTS содержат все такие описания. Кроме этого, дополнительные точные повторы появились при конвертации исходных текстов документов утилитой Pandoc из-за недокументированных особенностей утилиты Pandoc в случае конвертации TROFF-формата.

В документе Python Requests точные повторы присутствуют из-за наличия общих параметров в различных классах и методах.

Глава 3. Система по работе с эталонными коллекциями неточных повторов

3.1. Общее описание

Была создана система, позволяющая облегчить работу пользователя по редактированию и созданию коллекций, а также позволяющая проводить сравнительный анализ данной коллекции с результатом работы алгоритма, используя в качестве метрик точность и полноту. На рис. 1 представлена UML-диаграмма компонент системы, которая была создана с помощью сервиса creately.com [11].

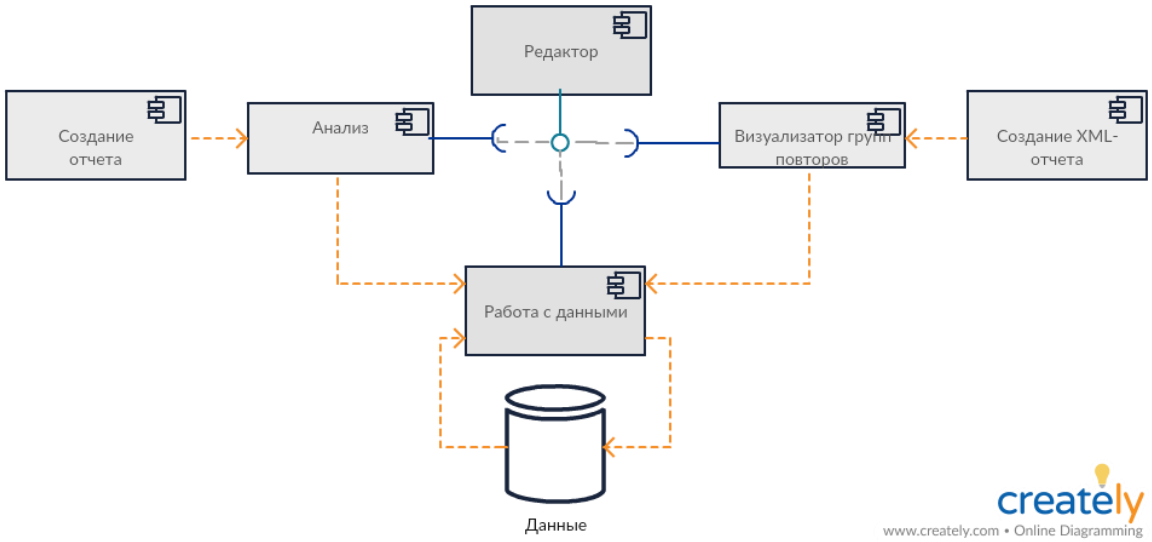


Рис. 1. Диаграмма компонент системы

Компонента “Редактор” позволяет отображать коллекцию и исходный документ, производить поиск фрагментов текста в документе, находить выбранный повтор из коллекции в документе, добавлять новые группы и повторы в коллекцию, а также удалять их.

Компонента “Визуализатор групп повторов” позволяет составлять XML-отчёт по текущей коллекции. Она отображает группы повторов и показывает различающиеся части каждого повтора.

Компонента “Анализ” сравнивает коллекцию и результат работы автоматического алгоритма Duplicate Finder и составляет отчёт данного сравнения.

Компонента “Работа с данными” выполняет чтение и преобразование исходных данных, которые хранятся в формате json, в словарь, с которым работает система, а также позволяет преобразовать данные из системы в формат json и записать их в файл.

3.2 Формат хранения данных

Данные, которые принимает на вход программа, представляют собой:

- фрагмент документации, представлен в формате txt;
- файл в формате json, содержащий эталонную коллекцию для выбранного фрагмента, его название: <Название документа, для которого представлена коллекция.txt.json>;
- файл в формате json, полученный при анализу документа алгоритмом Duplicate Finder, его название: <Название документа, для которого представлена коллекция.txt.report.json>.

3.3 Функционльность

Программа позволяет производить поиск фрагмента текста. При этом пользователь имеет возможность выделить выбранный фрагмент курсором и выбрать дальнейшее действие — добавить в существующую группу повторов, создать новую группу.

Начальное окно работы с программой представлено на рис. 2.

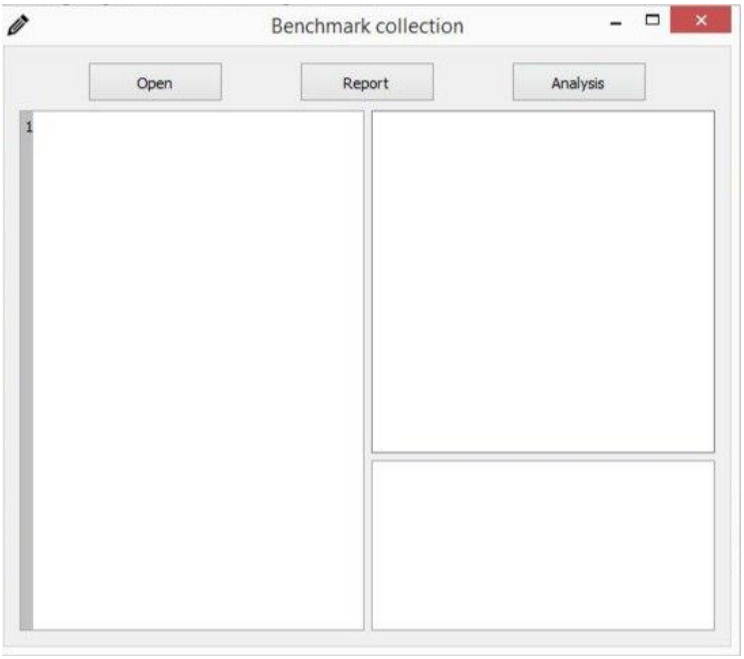


Рис. 2. Начальное окно программы

На рис. 3 представлен пример работы функции поиска.

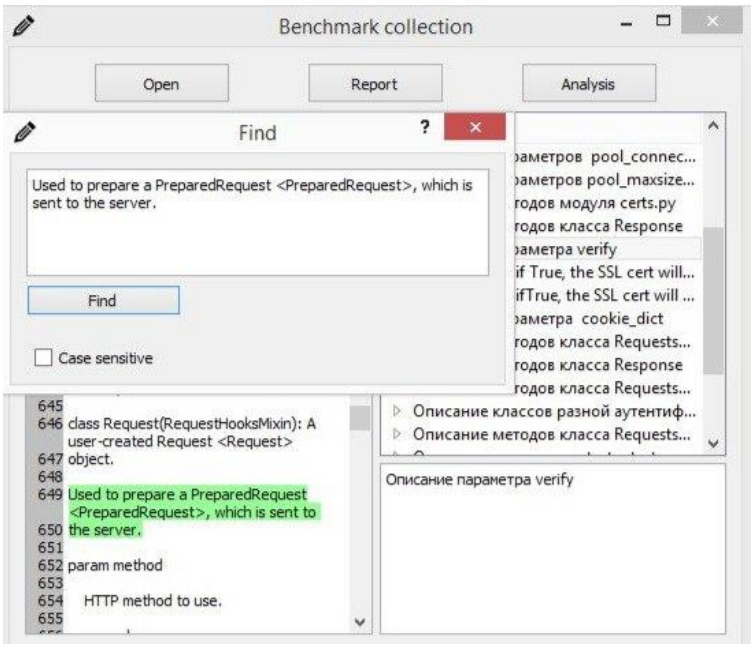


Рис. 3. Функция поиска

На рис. 4 представлен пример работы функции по добавлению новой группы повторов.

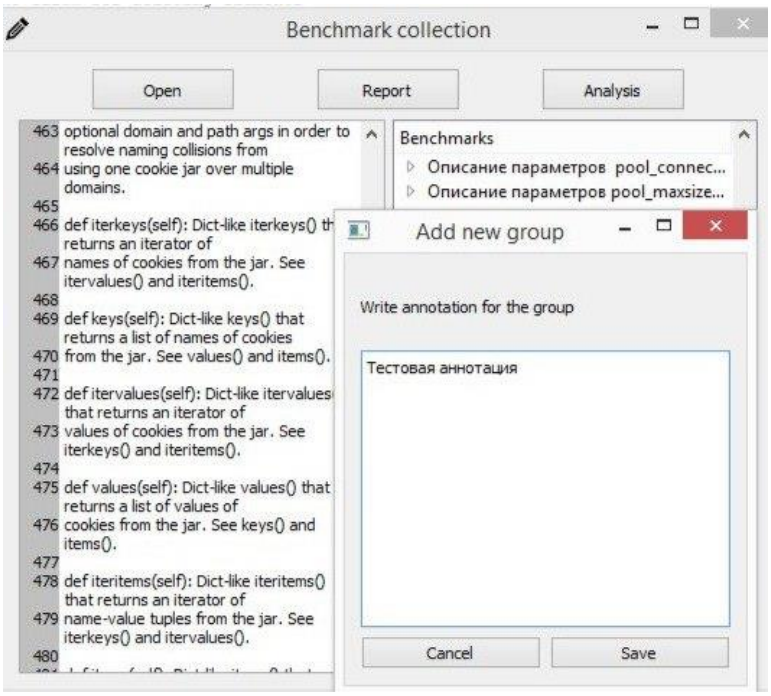


Рис. 4. Добавление новой группы в коллекцию

Если пользователь выберет один из повторов, программа отобразит его месторасположение в тексте документа, пример работы данной функции представлен на рис. 5.

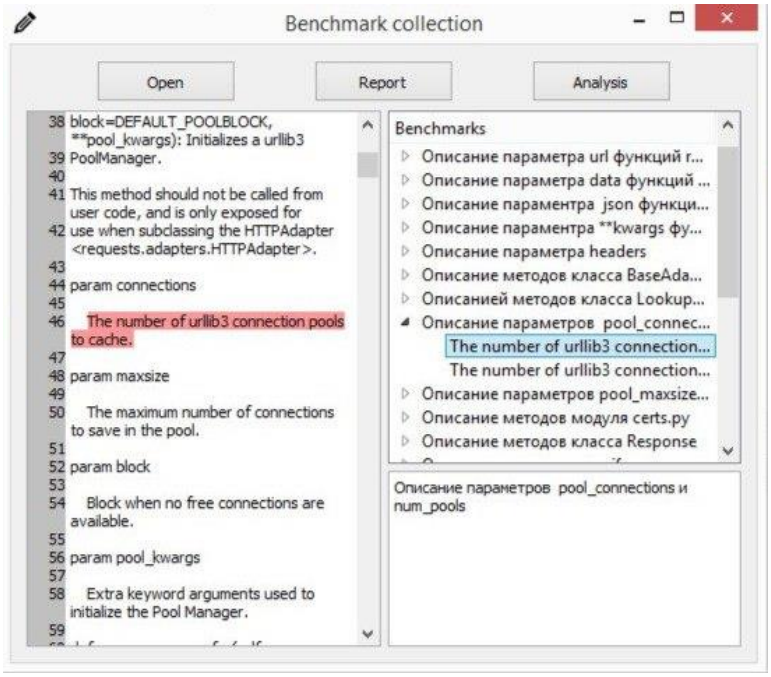
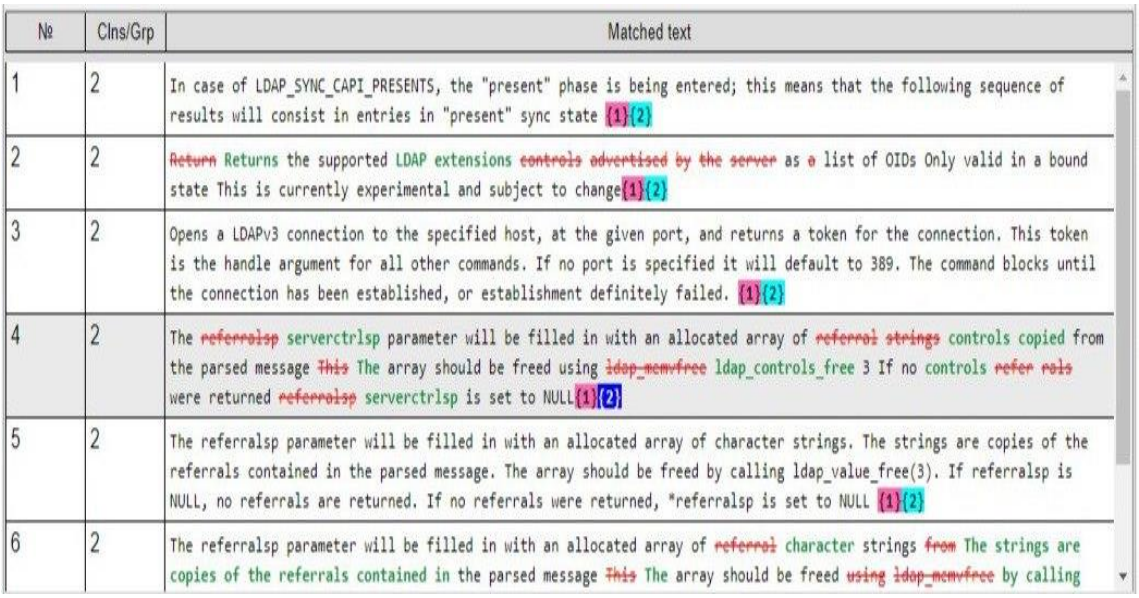


Рис. 5. Поиск выбранного повтора в документе

Также программа позволяет редактировать существующие группы: менять аннотацию к ним, удалять выбранные повторы или саму группу.

Для удобства пользователя имеется возможность отобразить коллекцию в браузере. Создаётся отчёт, показывающий количество повторов в группе, различия между ними, а также позволяющий находить выбранный повтор в тексте документа. Пример такого отчёта представлен на рис. 6.



№	Clns/Grp	Matched text
1	2	In case of LDAP_SYNC_CAPI_PRESENTS, the "present" phase is being entered; this means that the following sequence of results will consist in entries in "present" sync state {1}{2}
2	2	~~Return~~ Returns the supported LDAP extensions ~~controls advertised by the server~~ as ~~a~~ list of OIDs Only valid in a bound state This is currently experimental and subject to change{1}{2}
3	2	Opens a LDAPv3 connection to the specified host, at the given port, and returns a token for the connection. This token is the handle argument for all other commands. If no port is specified it will default to 389. The command blocks until the connection has been established, or establishment definitely failed. {1}{2}
4	2	The ~~referralsp~~ serverctrlsp parameter will be filled in with an allocated array of ~~referral strings~~ controls copied from the parsed message ~~This~~ The array should be freed using ~~ldap_memvfree~~ ldap_controls_free 3 If no controls ~~refer rols~~ were returned ~~referralsp~~ serverctrlsp is set to NULL{1}{2}
5	2	The referralsp parameter will be filled in with an allocated array of character strings. The strings are copies of the referrals contained in the parsed message. The array should be freed by calling ldap_value_free(3). If referralsp is NULL, no referrals are returned. If no referrals were returned, *referralsp is set to NULL {1}{2}
6	2	The referralsp parameter will be filled in with an allocated array of ~~referral~~ character strings ~~from~~ The strings are copies of the referrals contained in the parsed message ~~This~~ The array should be freed ~~using ldap_memvfree~~ by calling

Рис. 6. Пример HTML-отчёта

3.4 Способ оценки качества алгоритмов поиска повторов

Данная система создавалась для оценки качества алгоритмов автоматического поиска. Для оценки качества были выбраны следующие метрики:

- точность (Precision);
- полнота (Recall);
- средняя точность (Average Precision);
- средняя полнота (Average Recall).

Опишем эти метрики более подробно.

Метрики разделены на две группы. Первая группа (точность и полнота) рассматривает найденные повторы как цельные сущности и позволяет сказать, сколь полно совокупное множество этих повторов. Вторая группа

метрик (средняя точность и средняя полнота) позволяет проанализировать, сколь качественно был найден каждый повтор из коллекции, ведь алгоритм может найти лишь часть повтора, либо захватить лишние фрагменты текста.

Точность и полноту будем вычислять в соответствии с тем, насколько повторы из эталонной коллекции пересекаются повторами, найденными анализируемым алгоритмом. Идея этого сравнения представлена на рис. 7.

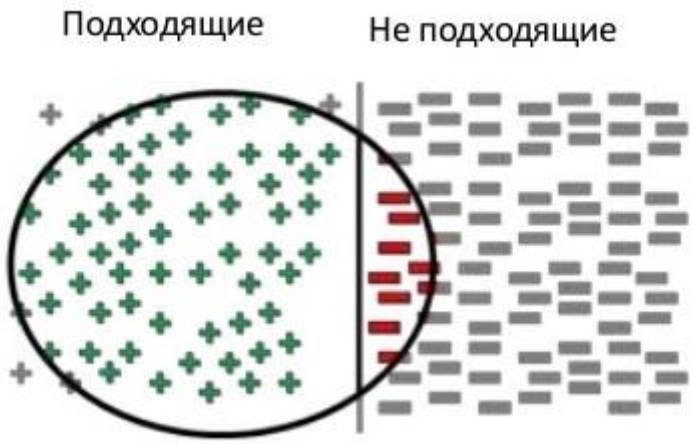


Рис. 7. Точность и полнота

На этом рисунке слева от прямой черты изображены повторы, которые содержатся в коллекции, а повторы, найденные алгоритмом, находятся в овале. Область красного цвета — это ошибки алгоритма. Таким образом *точность* (Precision) — это пропорция левой зелёной области по отношению ко всему овалу. Следуя [10], будем вычислять точность следующим образом:

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}},$$

где TP — это истинно-положительное решение, что в данном случае означает следующее: для каждого повтора из эталонной коллекции выясняется, пересекается ли он с различными повторами из выдачи алгоритма. Тогда TP является суммой элементов из эталонной коллекции, для которых утверждение о пересечении верно. FP — это ложно-положительное решение, то есть количество повторов из выдачи алгоритма, которые не пересеклись ни с одним повтором из коллекции.

Полнота (Recall) — это пропорция левой зелёной области рис. 7 ко всей области слева от прямой, и, следуя [10], она вычисляется так:

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}},$$

где FN — это ложно-отрицательное решение, то есть количество повторов, которые не были найдены алгоритмом, но при этом содержатся в коллекции.

Перейдем к метрикам второй группы, необходимым для проверки того, на сколько найденные анализируемым алгоритмом повторы, которые при этом содержатся также и в эталонной коллекции, соответствуют повторам из коллекции посимвольно. Идея этого сравнения представлена на рис. 8.

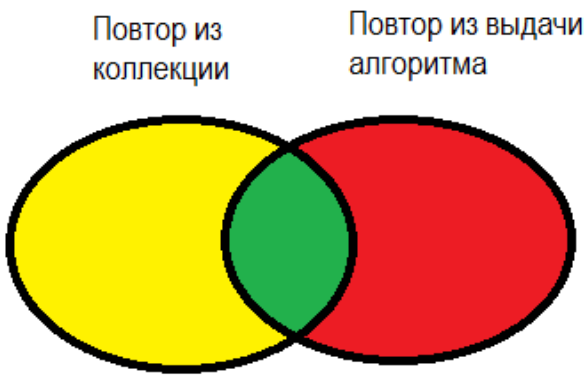


Рис. 8. Идея сравнения (посимвольно) двух повторов

На этом рисунке овал, изображенный слева, соответствует фрагменту текста, который содержится в повторе из эталонной коллекции. Овал, изображенный справа, соответствует фрагменту текста, который содержится в повторе из выдачи алгоритма. Область зеленого цвета — это количество текста (в символах), которое содержится в пересечении повторов. Область красного цвета — количество текста (в символах), содержащегося в повторе из выдачи, но не являющегося частью повтора из коллекции. Область желтого цвета — количество текста (в символах), содержащегося в повторе из эталонной коллекции, но не являющегося частью повтора, найденного

алгоритмом. Таким образом *средняя точность* (Average Precision) — это пропорция зелёной области по отношению к правому овалу. *Средняя полнота* (Average Recall) — это пропорция зеленой области по отношению к левому овалу. Средняя полнота показывает, насколько в среднем повторы из выдачи покрыли собой повторы из коллекции.

Существует два метода для вычисления этих метрик: micro-average и macro-average [13]. В первом методе суммируются отдельные истинно-положительные, ложно-положительные и ложно-отрицательные решения для всех повторов. Вычисления значений средней точности и средней полноты проводятся по формулам, данным ниже.

$$Average\ Precision\ = \frac{\sum_1^n TP_q}{\sum_1^n TP_q + \sum_1^n FP_q},$$

где n — это количество повторов, содержащихся в коллекции, которые были найдены алгоритмом, TP_q — это истинно-положительное решение, что в данном случае означает следующее: количество текста (в символах), полученного пересечением повтора из коллекции с повтором из отчета. FP_q — это ложно-положительное решение, то есть количество текста (в символах), принадлежащего повтору из выдачи, но не являющегося частью повтора из коллекции.

$$Average\ Recall\ = \frac{\sum_1^n TP_q}{\sum_1^n TP_q + \sum_1^n FN_q},$$

где $\mathrm{FN_q}$ — это количество текста (в символах), принадлежащего повтору из коллекции, но не содержащегося в повторе из выдачи.

В рамках данной работы повторы рассматриваются независимо, т.е. метрика должна характеризовать качество нахождения алгоритмом каждого отдельно взятого повтора, метод вычисления micro-average не позволяет дать такую оценку. В macro-average методе вычисляются значения метрик точности и полноты отдельно для каждой пары повторов (один повтор из эталонной коллекции, второй из выдачи алгоритма), а затем считается

среднее от полученных значений метрик. Вычисления средней точности и полноты проводились по следующим формулам:

$$Average\ Precision = \frac{1}{\mathrm{n}} \cdot (\sum_{q=1}^{n} P_q),$$

$$Average\ Recall = \frac{1}{\mathrm{n}} \cdot (\sum_{q=1}^{n} R_q),$$

где n — это количество повторов, содержащихся в коллекции, которые были найдены алгоритмом; q — это рассматриваемый повтор из коллекции; P_q — точность, соответствующая повтору q; R_q — полнота, соответствующая повтору q. При этом P_q и R_q вычисляются следующим образом:

$$P_q = \frac{\mathrm{TP_q}}{\mathrm{TP_q} + \mathrm{FP_q}},$$

$$R_q = \frac{\mathrm{TP_q}}{\mathrm{TP_q} + \mathrm{FN_q}},$$

где $\mathrm{TP_q}$ — это количество текста (в символах), полученного пересечением повтора из коллекции с повтором из отчета; $\mathrm{FP_q}$ — это количество текста (в символах), который принадлежит повтору из выдачи, но при этом не является частью повтора из коллекции; $\mathrm{FN_q}$ — это количество текста (в символах), которое принадлежит повтору из коллекции, но не содержится в повторе из выдачи.

При вычислении P_q и R_q может возникнуть ситуация, когда одному повтору из коллекции соответствует несколько повторов из выдачи алгоритма, поэтому дополнительно для каждой пары сравниваемых повторов вычисляется *F-мера* (F-mesure) [10] и в качестве итоговых значений для метрик P_q и R_q выбираются те, при которых значение F-меры максимально.

F-мера представляет собой гармоническое среднее между точностью и полнотой и вычисляется по следующей формуле:

$$F_q = 2 \cdot \frac{P_q \cdot R_q}{P_q + R_q}.$$

Значения P_q и R_q вычисляются для каждой пары пересекающихся повторов по формулам, определённым выше.

Глава 4. Апробация системы

Разработанная система была апробирована на созданных коллекциях. Использовался алгоритм компоновки [1] инструмента Duplicate Finder, который запускался на тех же документах, для которых строились эталонные коллекции — Python Requests и LibLDAP Manual.

4.1. Пример 1

В качестве перового примера взят документ Python Requests. В настройках автоматического режима Duplicate Finder (это и есть алгоритм компоновки) были выставлены следующие параметры: минимальная длина в токенах точных повторов, фиксируемых Clone Miner равна 1, минимально допустимая длина в токенах архетипа неточных групп равна 5.[3] На рис. 9 представлены данные сравнения эталонной коллекции и результатов, полученных алгоритмом компоновки.

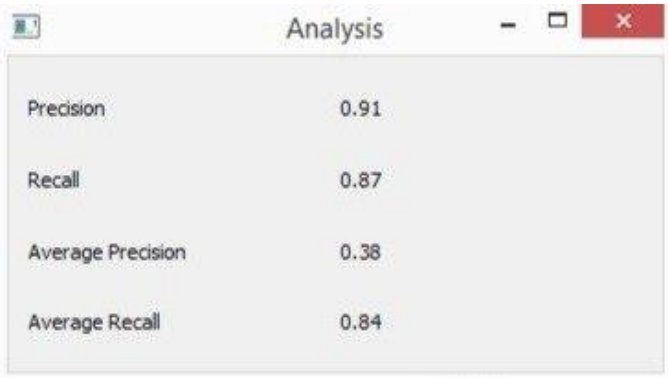


Рис. 9. Результат для Python Requests

Сравнение показало высокую точность (Precision = 0.91), и это означает, что алгоритм компоновки на документе Python Request выдает малое количество нерелевантных повторов. Это удобно для пользователя, который сразу получает желаемый результат. Высокое значение коэффициента полноты (Recall = 0.87) означает, что алгоритму удалось найти почти все релевантные повторы в анализируемом документе. Высокое значение коэффициента средней полноты (Average Recall = 0.84) показывает, что

[3] Обсуждение оптимальности этих параметров можно найти в [2].

повторы, найденные алгоритмом, почти полностью содержали в себе повторы из коллекции. Коэффициент средней точности (Average Precision = 0.38) имеет достаточно низкое значение, а значит, несмотря на то, что алгоритм нашел почти все релевантные повторы и также содержал их в себе, он не смог правильно выделить границы повторов. Пример этого утверждения представлен ниже.

```
   to send in the body of the Request. param headers (optional) Dictionary of HTTP Headers to
send with the Request. param cookies (optional) Dict or CookieJar object to send with the
Request. param files (optional) Dictionary of '◂1◂ name': | filename': ▸1▸file-like-objects ◂2
◂ (or
{'name': ('filename', fileobj)}) | ε ▸2▸for multipart encoding upload. param auth (optional) Auth
tuple ◂3◂ ε | or callable ▸3▸to enable Basic/Digest/Custom HTTP Auth. param timeout (optional) How
long to wait for the server to send data before giving up, as a float, or a (connect timeout,
read                                                                                timeout)
tuple.    :type    timeout:    float    or    tuple    :param    allow_redirects:    (optional)◂4◂
Boolean. | ε ▸4▸Set to True
```

В данном примере представлена группа повторов, найденная алгоритмом, цветом выделены различные точки расширения. В этом фрагменте текста описываются различные параметры (headers, cookies, files и т.д.), которые представляют собой различные программные сущности, поэтому в коллекции данные повторы находятся в разных группах. Если при рассмотрении результатов работы алгоритма опираться только на то, пересеклись ли повторы, то для данного случая все окажется верным, поэтому необходимо так же учитывать отношение пересекающегося фрагмента текста рассматриваемых повторов к их длинам (Average Precision и Average Recall), так как это покажет, насколько точно алгоритм выделяет границы.

4.2. Пример 2

В качестве второго тестового примера взят документ LibLDAP Manual. Настройки алгоритма компоновки были выставлены такими же, как и в предыдущем примере. На рис. 10 представлены данные сравнения эталонной коллекции и результатов, полученных автоматическим алгоритмом.

Рис. 10. Результат для LibLDAP Manual

Сравнение показало высокую точность (Precision = 0.84), это означает, что алгоритм компоновки на документе LibLDAP Manual выдает малое количество нерелевантных повторов. Коэффициент полноты (Recall), равный 0.50, означает, что алгоритм пропустил около половины релевантных повторов при анализе. Высокое значение коэффициента средней полноты (Average Recall = 0.93) показывает, что повторы, найденные алгоритмом, почти полностью содержали в себе повторы из коллекции. Коэффициент средней точности (Average Precision), равный значению 0.12, означает, что лишь малая часть повторов найдена с правильными границами. Такой низкий показатель при анализе документа LibLDAP Manual объясняется тем, что в документе содержится дублирование параграфов, описывающих функции и структуры данных. Эти параграфы включают в себя описания отдельных методов, параметров, разделов, и поэтому при сравнении возникает ситуация, когда повтор из коллекции, соответствующий описанию метода, будет содержаться в повторе из отчета, который содержит описание нескольких методов, соответственно, совпадёт лишь малая часть этих повторов. Ниже приведен пример фрагмента текста, содержащегося в коллекции, но отсутствующего в отчете.

The server certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, it will be ignored and the session proceeds normally.

….

The server certificate is requested. If no certificate is provided, the session proceeds normally. If a bad certificate is provided, the session is immediately terminated.

….

These keywords are equivalent. The server certificate is requested. If no certificate is provided**, or** a bad certificate is provided, the session is immediately terminated. **This is the default setting.**

В этом примере представлены описания ключевых слов для дополнительной опции конфигурации при построении OpenLDAP с TLS (Transport Layer Security). Зеленым выделена точка расширения, которая имеется у первого и второго фрагментов. При сравнении всех трех фрагментов видно, что при включении третьего в одну группу с остальными двумя, точки расширения у этих пар повторов будут отличаться. Жирным шрифтом выделены различия третьего повтора от предыдущих двух. Алгоритм компоновки выделил в качестве группы повторов только первые два описания, так как он объединяет повторы, у которых будут самые короткие (в символах) точки вариаций.

4.3. Вывод

Проведённая апробация показала, что алгоритм компоновки Duplicate Finder находит повторы в документации с высокой точностью, при этом найденные повторы почти полностью содержат в себе повторы из соответствующей эталонной коллекции. Опытным путем было установлено, что алгоритм не умеет выделять границы повторов, из-за этого в выдаче часто содержится избыток информации, либо присутствуют фрагменты, не несущие смысловой нагрузки. При этом значение метрики Recall было равно 0.87 и 0.50 для документов Python Requests и LibLDAP Manual соответственно. Это показало, что в первом случае алгоритм нашёл почти все релевантные повторы, а во втором он многое пропустил. Эта разница объясняется тем, что при поиске повторов в документе LibLDAP Manual алгоритм компоновки часто не включал в группы повторы, дублирующие описания методов и параметров. Например, описание LDAP_DECODER встречается в тексте 18 раз, т.е. получается одна группа, состоящая из 18

элементов, но несколько таких описаний идут в тексте документа друг за другом, поэтому алгоритм, который не умеет выделять семантические границы, объединил несколько подряд идущих описаний в один фрагмент и выделил в группу всего 2 повтора, содержащих по 4 подряд идущих описания. В результате, описания, стоящие раздельно, не были учтены.

Основываясь на полученных результатах, дальнейшим предметом анализа может стать метрика, показывающая насколько правильно алгоритм может выделять группы повторов: сколько повторов алгоритм поместил в одну группу ошибочно, сколько повторов были пропущены и не содержатся в группе. Результаты, полученные при анализе, и приведенные примеры, позволяют сказать, что алгоритм компоновки допускает ошибки при выделении групп. Это происходит из-за того, что он плохо выделяет границы повторов, тем самым «склеивая» несколько подряд идущих повторов в один, а также из-за того, что в одной группе могут содержаться повторы, имеющие различные точки расширения в зависимости от их попарной группировки друг с другом, и на данный момент алгоритм не учитывает эту особенность, группируя повторы на основе наименьшей длины вариативной части.

У предложенного в данной работе способа оценки есть недостаток, заключающийся в том, что минимальная длина пересечения повторов (в символах), при которой два повтора считаются пересекающимися, составляет один символ. Это значение не является достаточным для утверждения того, что повторы действительно пересеклись.

Заключение

В ходе выполнения данной работы были достигнуты следующие результаты:

1. Было выполнено ознакомление с методом работы инструмента Duplicate Finder. Изучены правила построения эталонных коллекций в информационном поиске.
2. Была изучена API-документация ПО, а именно, документы LibLDAP Manual, Python Requests и Zend Framework на предмет поиска и анализа неточных повторов. Для первых двух документов были созданы эталонные коллекции неточных повторов, включающие 76 и 31 группу соответственно.
3. Была спроектирована и реализована система для работы с эталонными коллекциями неточных повторов на языке Phyton в контексте инструментария Duplicate Finder.
4. Апробация созданного подхода и разработанной системы была выполнена на двух созданных коллекциях и алгоритме компоновки, реализованном в Duplicate Finder. Алгоритм запускался на документах LibLDAPManual и PythonRequests, для которых создавались эталонные коллекции. Тестирование показало, что почти все повторы, содержащиеся в выдаче алгоритма, являются релевантными, но алгоритм компоновки часто некорректно выделяет границы найденных повторов. Кроме этого алгоритм «склеивает» фрагменты текста, находящиеся рядом, но описывающие разные программные сущности (software features), в результате чего теряется большое количество релевантных повторов (метрика Recall = 0.50 для документа LibLDAPManual). Так же алгоритм компоновки не выделяет в одну группу повторы, которые имеют попарно различные точки расширения.

Список литературы

[1] Duplicate Finder [Electronic resource]. — URL: http://www.math.spbu.ru/user/kromanovsky/docline/index.html (online; accessed: 2018.04.20).

[2] Луцив, Д.В. Поиск неточных повторов в документации программного обеспечения / Д.В. Луцив. Диссертация на соискание научной степени кандидата физико-математических наук. — Санкт-Петербургский государственный университет. — 2018.

[3] Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. What is Data Quality and Why Should We Care? // Springer New York, New York, NY. — 2007. — P. 7–15.

[4] Wang, R. Y., Strong, D. M. Beyond Accuracy: What Data Quality Means to Data Consumers // Journal of Management Information Systems 12(4). — 1996. — P. 5–33.

[5] Nicola Askham, Denise Cook, Martin Doyle, Helen Fereday, Mike Gibson, Ulrich Landbeck, Rob Lee, Chris Maynard, Gary Palmer, and Julian Schwarzenbach. The six primary dimensions for data quality assessment // Technical report, DAMA UK Working Group. — 2013.

[6] Zhu X., Gauch S. Incorporating quality metrics in centralized/distributed information retrieval on the World Wide Web // Procedures of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Athens. ACM Press — 2000. — P. 288–295.

[7] Basit, H.A. Efficient Token Based Clone Detection with Flexible Tokenization / H. A. Basit, S. Jarzabek // Proceedings of the 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers. — 2007. — P.513–516.

[8] Cleverdon C.W. The significance of the Cranfield tests on index languages // Proceedings of 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Chicago, IL, USA. ACM Press — 1991. — P. 3–12.

[9] Chadwick D. Understanding X.500: The Directory // Chapman & Hall, Ltd. London, UK, UK — 1994.

[10] Manning C.D., Raghavan P., Schütze H. Introduction to Information Retrieval // Cambridge University Press — 2008. — P. 168–69.

[11] Creately: Diagram Maker [Electronic resource]. — URL: https://creately.com (online; accessed: 2018.04.22).

[12] David D. Lewis, Evaluating Text Categorization // HLT '91 Proceedings of the workshop on Speech and Natural Language — 1991. — P. 312–318.

[13] Vincent Van Asch, Macro- and micro-averaged evaluation measures [Electronic resource]. URL: https://www.clips.uantwerpen.be/~vincent/pdf/microaverage.pdf (online; accessed: 2018.05.01).

[14] MacFarlane, J. Pandoc: an universal document converter [Electronic resource] / J. MacFarlane. — UPL: http://pandoc.org (online; accessed: 2018.04.22)

Приложение. Примеры группы неточных повторов из созданных эталонных коллекций

В данном приложении представлены примеры групп повторов, содержащиеся в эталонной коллекции.

Пример 1

Номер	Повтор	Возможность
1	The requested **time** limit (in **seconds**); by default 0, to indicate no limit.	Int ls_timelimit
2	The requested **size** limit (in **entries**); by default 0, to indicate no limit.	Int ls_sizelimit

Таблица П1. Пример 1: описание параметров структуры ldap_sync_t

Это два сходных фрагмента документации LibLDAP Manual. Оба являются описанием параметров структуры ldap_sync_t, которая используется для процедур синхронизации LDAP. Различающиеся части выделены жирным шрифтом. Оба параметра задают запрашиваемые значения, поэтому их описания похожи, но один параметр отвечает за ограничение времени синхронизации, а другой за ограничение размера передаваемого сообщения.

Номер	Повтор	Возможность
1	These routines provide a subroutine interface to a simplified implementation of the Basic Encoding Rules of ASN.1. The version of BER these routines support is the one defined for the LDAP protocol. The encoding rules are the same as BER, except that only definite form lengths are used, and bitstrings and octet strings are always encoded in primitive form. This man page describes the **encoding** routines in the lber library. See lber-**decode(3)** for details on the corresponding **decoding** routines. Consult lber-types(3) for information	LBER_ENCODE

	about types, allocators, and deallocators. Normally, the only routines that need to be called by an application are **ber_alloc_t() to allocate a BER element for encoding, ber_printf()** to do the actual **encoding, and ber_flush2() to actually write the element**. The other routines are provided for those applications that need more control than **ber_printf()** provides. In general, these routines return the **length** of the element **encoded, or -1** if an error occurred.	
2	These routines provide a subroutine interface to a simplified implementation of the Basic Encoding Rules of ASN.1. The version of BER these routines support is the one defined for the LDAP protocol. The encoding rules are the same as BER, except that only definite form lengths are used, and bitstrings and octet strings are always encoded in primitive form. This man page describes the **decoding** routines in the lber library. See lber-**encode(3)** for details on the corresponding **encoding** routines. Consult lber-types(3) for information about types, allocators, and deallocators. Normally, the only routines that need to be called by an application are **ber_get_next() to get the next BER element and ber_scanf()** to do the actual **decoding. In some cases, ber_peek_tag() may also need to be called in normal usage**. The other routines are provided for those applications that need more control than **ber_scanf()** provides. In general, these routines return the **tag** of the element **decoded, or LBER_ERROR** if an error occurred.	LBER_DECODE

Таблица П2. Пример 2: базовые правила кодирования

Это два сходных фрагмента документации LibLDAP Manual. Оба фрагмента являются общим описанием параграфов библиотеки LDAP. Эти параграфы содержат информацию о функциях и структурах данных,

предназначенных для упрощенной реализации базовых правил кодирования Basic Encoding Rules (Basic Encoding Rules, BER). Различающиеся части текста выделены жирным шрифтом. В первом абзаце отличается лишь небольшая фрагментов, это объясняется тем, что LBER_DECODE отвечает за декодирование сообщений, а LBER_ENCODER — за кодирование. Сами правила кодирования для них общие. Отличия во втором абзаце возникают по той же причине: для кодирования и декодирования используются разные функции, которые выполняют обратные действия по отношению друг к другу.