Добрый день, меня зовут Глазырин Антон, и тема моей работы — универсальный механизм первичного поиска повторов в тексте для пакета Duplicate Finder.

Несколько слов про повторы в документации. Существует ряд исследований, которые показывают, что в среднем достаточно значительная часть документации крупных проектов — около 10% — представляет собой дублированные фрагменты. Хотя повторы в документации сами по себе не являются чем-то плохим, они могут приводить к некоторым негативным последствиям, таким как раздувание объема документации и усложнение внесения изменений, что затрудняет сопровождение. Кроме того, существуют различные методы по улучшению документации, основанные на управлении повторами.

Один из таких методов лежит в основе пакета Duplicate Finder. Этот инструмент является университетской разработкой и предоставляет функционал для улучшения документации за счет поиска повторов. Из-за того, что данный проект разрабатывался и дополнялся в течение длительного промежутка времени многими разными людьми, у него появился ряд проблем с компонентами поиска. Дело в том, что в Duplicate Finder нет своего механизма поиска повторов: для этого используются набор внешних инструментов. Однако, эти инструменты написаны на разных языках, больше не поддерживаются, а самый важный из них — CloneMiner — является закрытой разработкой.

Таким образом, целью данной работы является разработка унифицированной подсистемы поиска точных и неточных повторов для Duplicate Finder Toolkit. Для достижения данной цели были поставлены следующие задачи: ...

Поиск повторов очень широко распространен в различных сферах, и часто инструменты достаточно сильно заточны под свою конкретную задачу. Среди областей применения наиболее выделяются следующие. Это поиск клонов в исходном коде ПО: такие инструменты как CCFinder или CloneMiner, который как раз и используется в Duplicate Finder, сравнение текстовых документов: самый яркий пример — это проверка на плагиат: такие инструменты как Align и TxtAlign, и поиск похожих вхождений по образцу: например, библиотека для поиска Apache Lucene.

На основе анализа проблем поиска повторов в Duplicate Finder определены следующие требования к новому механизму: инструмент должен быть реализован на языке Python для максимальной совместимости с Duplicate Finder, быть открытой разработкой, и иметь возможности как точного так и неточного поиска повторов, сам процесс поиска должен быть универсальным, необходимо наличие API и CLI и должна быть возможность настройки параметров алгоритмов поиска.

На данном слайде представлена схема разработанного конвейера поиска повторов. Он включает в себя 3 основных этапа: предобработку текста для улучшения качества поиска, непосредственно применение алгоритмов поиска повторов, и затем балансировку полученных групп с целью улучшить качество результатов. Далее рассмотрим каждый этап подробнее.

Первым шагом является предобработка, которая выполняет 2 задачи: во-первых — преобразование текста в удобный для дальнейшей работы вид, во-вторых — применение ряда методов для облегчения работы алгоритмов поиска. Конкретно — это подходы, широко используемые в NLP — обработке естественного языка. Они включают в себя устранение шума — фильтрацию спецсимволов и удаление стоп слов, а также приведение слов к наиболее унифицированной форме при помощи лемматизаци и стемминга. Кроме того текст токенизируется, и в дальнейшем представляется в виде набора токенов.

Далее рассмотрим алгоритмы поиска повторов. Всего были разработаны 3 алгоритма на основе компонент из Duplicate Finder: один для точного поиска и два для неточного. Алгоритм точного поиска основывается на построении суффиксного массива, который представляет собой последовательность суффиксов строки, упорядоченных в лексиграфическом порядке. Суть алгоритма заключается в том, что если в тексте содержатся два одинаковых фрагмента, то суффиксы, начинающиеся с этих фрагментов, будут соседями с суффиксном массиве, таким образом анализируя его можно находить и объединять повторы в группы.

Первый алгоритм неточного поиска основан на вычисление расстояния Левинштейна между фрагментами — количество операций удаления, замены и вставки нужное, чтобы превратить одну строку в другую. Если расстояние достаточно маленькое — значит фрагменты похожи и являются неточными повторами. Однако вычисление расстояния Левинштейна достаточно трудоемкий процесс, и не имеет смысла проводить этот расчет для сильно различающихся фрагментов.

Для грубой оценки схожести можно использовать хеширование, в частности — подход SimHash, который позволяет определить хеш коллекции на основе хешей ее составных частей, и затем использовать его для сравнения. Таким образом, фрагменты, имеющие много общих токенов, будут иметь схожие хеши. Суть алгоритма заключается в разбиение текста на фрагменты и попарного их сравнения при помощи этих двух подходов, с последующим объединением найденный пар в группы.

Второй алгоритм неточного поиска основан на еще одном распространенном подходе для оценки схожести текстов — построении множества N-грам. Имея два фрагмента можно построить их множества и вычислить пересечение, и чем ближе мощность полученного множества к исходному, тем больше эти фрагменты имеют одинаковых текстовых участков. Суть алгоритма заключается в разбиении текста на предложения и объединение их в группы повторов на основе вычисления пересечений их множеств N-грам.

Последним этапом конвейера является балансировка групп повторов. Он включает в себя удаление незначимых групп — которые содержат меньше 2 фрагментов, и слияние фрагментов — если есть группы, все фрагменты которых находятся в исходном тексте рядом с фрагментами другой группы, можно перераспределить эти фрагменты между группами и объединить их, что повышает значимость результатов поиска.

На данном слайде приведена архитектура реализованного инструмента. За каждый этап конвейера отвечает своя компонента, что позволят легко влиять на процесс поиска и добавлять новые алгоритмы.

Для тестирования были выбраны документации ряда различных крупных проектов. Для каждого документа был выполнен поиск точных и неточных повторов. Результаты приведены на слайде: неточный поиск...

... и точный поиск. Как видно по результатам, инструмент хорошо находит как точные, так и неточные повторы. Кроме того, на практике подтверждается теоретические ожидания: примерно 10% документа составляют дублированные участки. Исключением является Python Requests, которая представляет собой API-документацию и содержит много однообразного кода и описаний. Это также отражает идею, что в некоторых случаях повторы в документации неизбежны и выполняют определенную задачу.

Также была проведена интеграция реализованного инструмента в DuplicateFinder и его сравнение с основным используемым там инструментом CloneMiner. На данном графике показано сколько групп повторов было найдено для документов при использовании каждого инструмента отельно от Duplicate Finder’a.

А на этом — в результате поиска повторов с последующей обработкой DuplicateFinder’ом. Количество групп CloneMiner’а уменьшилось заметно сильнее. Это происходит потому, что CloneMiner создает много групп с пересечениями, которые отфильтровываются при обработке как незначимые.

Например, в случае если есть несколько фрагментов, образующих группу повторов, при этом у некоторых из них имеются продолжения, которые совпадают, CloneMiner создаст еще одну группу не в зависимости от их размера, ...

...которая затем будет отфильтрована DuplicateFinder’ом. Разработанный инструмент при этом всегда создает группы более логично и без пересечений, из-за чего результаты получаются более осмысленные и DuplicateFinder отсеивает гораздо меньше групп.

Таким образом, были достигнуты следующие результаты.