МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**

|  |  |  |
| --- | --- | --- |
|  | | |
|  | **Институт  интеллектуальных кибернетических систем** | |
| **Кафедра №22 «Кибернетика»** | |
| Направление подготовки 01.03.02 Прикладная математика и информатика | | |
| **Пояснительная записка** | | |
| к учебно-исследовательской работе студента на тему: | | |

Разработка android приложения для распознавания текста с включением нечёткого поиска

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Группа | | Б17-501 | | | |  | | | |
| Студент | |  | | | |  | Борзенков А.В. | | |
|  | | (подпись) | | | | (ФИО) | | | |
| Руководитель | |  | | | |  | Козин Р.Г. | | |
|  | | (подпись) | | | | (ФИО) | | | |
| Научный консультант | |  | | | |  |  | | |
|  | | (подпись) | | | | (ФИО) | | | |
| Оценка руководителя |  | | Оценка консультанта | | | | |  | |
|  | (0-100 баллов) | |  | | | | | (0-100 баллов) | |
|  |  | |  | | | | |  | |
| Итоговая оценка |  | | ECTS | | | | |  | |
|  | (0-100 баллов) | |  | | | | |  | |
| Комиссия | | | | | | | | | |
| Председатель |  | | |  |  | | | |  |
|  | (подпись) | | |  | (ФИО) | | | |  |
|  |  | | |  |  | | | |  |
|  | (подпись) | | |  | (ФИО) | | | |  |
|  |  | | |  |  | | | |  |
|  | (подпись) | | |  | (ФИО) | | | |  |
|  |  | | |  |  | | | |  |
|  | (подпись) | | |  | (ФИО) | | | |  |

**Москва 2020**

**Реферат**

Пояснительная записка содержит N страниц, M рисунков.

Количество использованных источников – 10.

Ключевые слова – android приложение, нечёткий поиск, метрика.

Целью данной работы является разработка метода увеличения точности с использованием нечёткого поиска и разработка android приложения для метода распознавания русского рукописного текста, разработанного ранее.

В первом разделе рассматриваются существующие алгоритмы увеличения точности методов распознавания текста, метод нечёткого поиска для увеличения точности определения распознанного слова. А также рассматриваются подходы и методологии по проектированию android приложения.

Во втором разделе производится разработка метода нечёткого поиска, а также определяется архитектура и методология для последующей реализации android-приложения.

В третьем разделе производится полный процесс реализации android-приложения.

В заключении производится тестирование разработанного приложения и подводятся итоги по каждому разделу.

**Содержание**

Введение1

Раздел 1. Анализ существующих подходов к оптимизации методов распознавания и проектированию android приложения1

1.1 Изучение и сравнительный анализ методов увеличения точности метода распознавания2

1.2 Изучение и сравнительный анализ методов нечёткого поиска2

1.3 Изучение существующих методологий и способов проектирования android приложения2

1.4 Выводы2

1.5 Постановка задачи курсового проекта2

Раздел 2. Разработка метода с использованием нечёткого поиска и архитектуры android приложения1

2.1 Выбор алгоритмов увеличения точности распознавания 2

2.2 Выбор методологии для проектировки android приложения2

2.3 Выводы2

Раздел 3. Разработка android приложения и метода оптимизации1

3.1 Выбор языка программирования для реализации android приложения2

3.2 Программная реализация метода оптимизации и android приложения2

3.3 Выводы2

Раздел 4. Экспериментальные исследования android приложения и метода нечёткого поиска1

4.1 Тестирование разработанного приложения2

4.2 Выводы по результатам тестирования2

**Введение**

Нечеткий поиск – это поиск информации, при котором выполняется сопоставление информации заданному образцу поиска или близкому к этому образцу значению. Другими словами, нечёткий поиск – это алгоритм поиска для данной подстроки другой такой подстроки в тексте, которая является наиболее близкой к данной по какой-либо определённой метрике. Алгоритмы нечеткого поиска используются в большинстве современных поисковых систем, например, для проверки орфографии. Для оптимизации разработанного ранее нейросетевого метода распознавания русского рукописного текста требуется произвести оптимизацию для увеличения точности распознавания. Используемый в работе метод оптимизации точности распознавания предполагает наличие в методе нечёткого поиска. Также для увеличения комфорта использования метода распознавания русского рукописного текста будет разработано android-приложение, позволяющее использовать данный метод.

В рамках работы решаются следующие задачи:

* Изучение существующих методов нечёткого поиска
* Изучение существующих архитектур и методологий для разработки android приложения
* Программная реализация нечёткого поиска
* Разработка android-приложения с использованием нечёткого поиск

**Раздел 1.** **Анализ существующих подходов к оптимизации методов распознавания и проектированию android приложения.**

Для увеличения точности поставленной задачи распознавания русского рукописного текста требуется определить различные способы оптимизации распознавания, которые могут быть применимы в данном контексте.

Для увеличения комфортности использования метода распознавания русского рукописного текста можно разработать android приложение.

В данной работе будет рассмотрен алгоритм для оптимизации точности распознавания русского рукописного текста, а также полный процесс разработки android приложения для реализации использования данного метода распознавая.

**1.1 Анализ существующих подходов к оптимизации методов распознавания и проектированию android приложения.**

Существует несколько различных методов для увеличения точности распознавания текста. Ниже приведены некоторые из них:

* Нечёткий поиск по словарю
* Лексический и грамматический анализы
* Методы, использующие нейронные сети
* Методы, использующие скрытые Марковские модели

Нечёткий поиск (fuzzy string search) это метод поиска наиболее подходящего слова в словаре к данному. Ранее разработанный алгоритм для каждого слова и для каждой распознанной буквы давал вектор вероятностей, состоящий из 51ого элемента, каждый из которых характеризовал то, на сколько данный символ «схож» с i-ым классом:

* Все строчные буквы русского алфавита за исключением буквы ‘ё’ (32 шт.)
* Все цифры (10 шт.)
* Прописные буквы русского алфавита, которые значимо не совпадают

визуально со строчными – ‘А’, ‘Б’, ‘В’, ‘Г’, ‘Д’, ‘П’, ‘Р’, ‘T’, ‘У’. (9 шт.)

Для слова из k-символов на выходе нейросетевого метода выдавался двумерный массив размерности [k, 51], который соответствовал векторам-вероятностям для каждого из рассматриваемого символа.

Поиск оптимального подходящего под результат символа сводился к выбору одного из 51 класса с наибольшей вероятностью, что не рассматривало зависимость символа от других символов в данном слове.

Для общей оптимизации метода распознавания можно использовать в данном процессе нечёткий поиск по словарю для каждого слова из текста, чтобы определить наиболее похожее слово из общего словаря. Такой подход может значимо минимизировать погрешности распознавания, используя различные корреляционные моменты между символами в слове.

Методы, основанные на лексическом и грамматическом анализе, методы использующие нейронные сети, а также методы с использованием скрытых Марковских моделей для увеличения точности распознавания текста подразумевают намного более комплексный подход, сводящийся к выделению смыслового контекста в предложениях и словах, а как следствие, во всём тексте сразу. Использование данных оптимизаций приведёт к значительному увеличению точности распознавания, однако, из-за отсутствия размеченной выборки хотя бы значимой части слов русского языка, данные методы не реализуемы, поэтому не могут быть использованных в дальнейшей работе.

**1.2 Изучение и сравнительный анализ методов нечёткого поиска.**

При рассмотрении методов распознавания рукописного текста с использованием нечёткого поиска необходимо выделить следующий ряд метрик, которые могут быть использованы в работе:

1. Расстояние Хэмминга
2. Расстояние Левенштейна
3. Расстояние Дамерау-Левенштейна

Расстояние Хэмминга – число позиций, в которых соответствующие символы двух слов одинаковой длины различны. В более общем случае расстояние Хэмминга применяется для строк одинаковой длины любых q-ичных алфавитов и служит метрикой различия объектов одинаковой размерности. Ниже изображен пример подсчёта расстояния Хэмминга для двух слов одинаковой длины (Рис. 1). Операция M – Math – сопоставление символов, операция R – Replace – замещение символов. Для заданного слова, различающиеся с исходным словом позиции окрашены красным цветом. Таким образом, расстояние Хэмминга между данными словами равно трём.

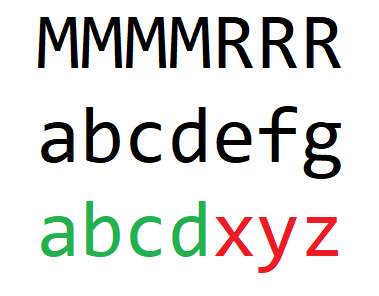


Рис 1 – Пример расстояния Хэмминга для двух слов

Расстояние Левенштейна (редакционное состояние, дистанция редактирования) – метрика, измеряющая разность между двумя последовательностями символов. Она определяется как минимальное количество односимвольных операций (а именно вставки, удаления, замещения), необходимых для превращения одной из последовательности символов в другую. В общем случае, операциям, используемым в этом преобразовании, можно назначить разные цены. Пример нахождения расстояния Левенштейна для двух слов различной длинны изображён ниже. Для обозначения операций используются следующие обозначения:

1. M – Match – операция совпадения двух символов
2. I – Insert – операция вставки нового символа
3. R – Replace – операция замещения одного символа другим
4. D – Delete – операция удаления символа
5. S – Swap – операция транспозиции (используется для модификаций алгоритма)

Для нахождения расстояния Левенштейна можно использовать алгоритм Вагнера-Фишера, основанный на динамическом программировании. Расстояние Левенштейна для двух префиксов заданных строк (S1 и S2) длины n и длины m обозначим как L (n, m). Оно находится следующим образом:

Где m (a, b) равно единице, если символы a и b различны, и 0 в противном случае.

Рассмотрим (Рис 2) пример нахождения расстояния Левенштейна для двух строк различной длины. Возьмём S1 := ‘abacabaa’, S2 := ‘badcba’

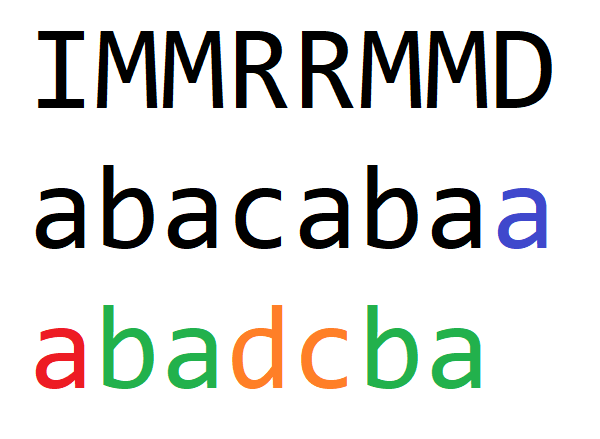


Рис 2 – Пример поиска расстояния Левенштейна

Красным обозначены символы, которые были добавлены (операция Insert). Зелёным обозначены совпадающие символы (операция Math). Оранжевым обозначены символы, которые были заменены (операция Replace). Синим обозначены символы, которые были удалены (операция Delete). Таким образом, расстояние Левенштейна между данными словами равно четырём. Ниже приведена таблица подсчёта для данных слов расстояния Левенштейна по алгоритму Вагнера-Фишера (Рис 3).

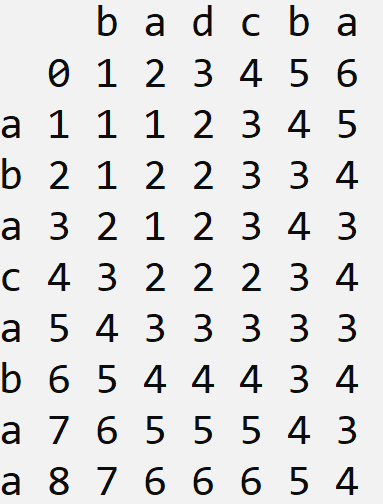


Рис 3 – Пример результата работы алгоритма Вагнера-Фишера

Расстояние Дамерау-Левенштейна (названо в честь учёных Фредерика Дамерау и Владимира Левенштейна) – это мера разницы двух строк символов, определяемая как минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов) необходимых для перевода одной строки в другую. Является модификацией расстояния Левенштейна. Расстояние Дамерау-Левенштейна для двух префиксов заданных строк (S1 и S2) длины n и длины m обозначим как DL (n, m). Оно находится следующим образом:

Где m (a, b) равно единице, если символы a и b различны, и 0 в противном случае.

Рассмотрим (Рис 4) пример нахождения расстояния Дамерау-Левенштейна для двух строк различной длины. Возьмём S1 := ‘abacabaa' , S2 := ‘baacaba’

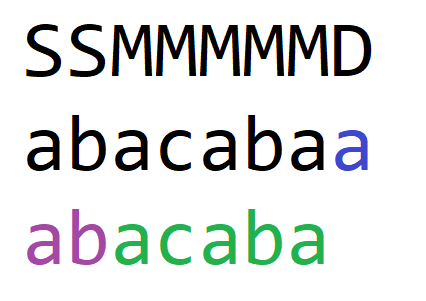


Рис 4 – Пример поиска расстояния Дамерау-Левенштейна

Зелёным обозначены совпадающие символы (операция Match). Синим обозначены символы, которые были удалены (операция Delete). Фиолетовым обозначена операция транспозиции (операция Swap). Таким образом, расстояние Дамерау-Левенштейна между данными словами равно трём. Ниже приведена таблица подсчёта для данных слов расстояния Левенштейна по алгоритму, основанному на динамическом программировании (Рис 5).

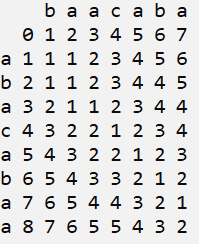


Рис 5 – Пример результата работы алгоритма Вагнера-Фишера

Все реализации выше использованных алгоритмов есть в приложении

Рассмотренные расстояния и алгоритмы для их нахождений, могут быть представленные в виде таблицы (Таблица 1):

*Таблица 1. Сравнение расстояний между символьными последовательностями*

|  |  |  |
| --- | --- | --- |
| Расстояние | Плюсы | Минусы |
| Хэмминга | Работает асимптотически за O(n), где n – длина сравниваемых строк | Не работает для строк различной длинны, также является примитивным алгоритмом посимвольного сравнения, не дающего полного анализа схожести двух слов |
| Левенштейна | Эффективно ищет всевозможные сочетания символов и находит «редакционное расстояние» | Работает асимптотически за O (n \* m), где n, m – длины сравниваемых строк |
| Дамерау-Левенштейна | Эффективно ищет всевозможные сочетания символов и находит «редакционное расстояние», также используя возможность транспозиции двух соседних символов. | Работает асимптотически за O (n \* m), где n, m – длины сравниваемых строк |

**1.3 Изучение существующих методологий и способов проектирования android приложения.**

Для разработки android-приложения требуется выбрать наиболее оптимальный способ проектирования приложения. Рассматривая официальную документацию android-разработчиков[1], следует выделить следующие аспекты каноничной и правильной проектировки приложения:

1. Приложение состоит из компонентов
2. Каждый компонент не должен зависеть от других компонентов, так как, в силу специфики строения android архитектуры, это может повлечь критические ошибки приложения
3. Ни один компонент не должен оперировать большим количеством данных, а также, не должен выполнять сложных вычислений, в силу крайне ограниченных мощностей мобильных телефонов
4. Пользователь должен управлять пользовательским интерфейсом из постоянной модели

Пример готовой архитектуры приложения представлен ниже (Рис 6). На нём отображены важнейшие компоненты архитектуры android приложения:

1. Activity / Framgent
2. ViewMode
3. Repository
4. Model
5. Remote Data Source



Рис 6 – Пример архитектуры приложения android

Рассматривая данную архитектуру, можно прийти к выводу, что общая специфика разработки приложения на android сводится к разделению и строгой иерархии всех отдельных частей приложения с целью увеличения отказоустойчивости и упрощения разработки программного приложения. Именно данные принципы и будут использоваться для разработки архитектуры android приложения.

**1.4 Выводы**

Оптимизация метода распознавания текста может быть выполнена несколькими различными способами – с использованием нечёткого поиска, нейронных сетей, скрытых Марковских моделей или с использованием различного вида анализов. В силу специфики разрабатываемого метода было решено выбрать метод оптимизации с использованием нечёткого поиска. Для архитектуры android приложения был выбран подход, использующий все основные принципы разработки канонического android-приложения.

**1.5 Постановка задач курсового проекта**

Целью данной работы является увеличение точности метода распознавания русского рукописного текста, а также разработка android приложения для увеличения применимости данного метода. Для достижения поставленных целей предполагается решить следующий ряд задач:

* Изучение существующих методов нечёткого поиска
* Изучение существующих архитектур и методологий для разработки android приложения
* Программная реализация нечёткого поиска
* Разработка android-приложения с использованием нечёткого поиск

**Раздел 2. Разработка метода с использованием нечёткого поиска и архитектуры android приложения**

В данном разделе рассматривается процесс разработки метода для улучшения качества распознавания русского рукописного текста с использованием нечёткого поиска, а также разработка архитектуры android приложения для использования данного метода.

**2.1 Выбор алгоритмов увеличения точности распознавания**

Ранее выбранный метод распознавания с использованием нечёткого поиска может быть реализован для различных метрик – каждая из которых характеризуется сложностью алгоритма вычисления и набором характеристик, определяющих схожесть слов.

Для использования любого из рассматриваемых методов требуется наличие списка из всех словоформ русского языка (или, по крайней мере, большей части его использующихся).

Один из подходящих вариантов содержит 173000 слов, и, соответственно более двух миллионов словоформ, что даёт почти полное покрытие всевозможных форм слов в русском языке[1]. Общее количество символов в словаре боле 26 миллионов, что вынуждает рассматривать только оптимальные для использования алгоритмы.

Был проведён сравнительный анализ скоростей работы алгоритмов для нечёткого поиска. Для заданной длины слова определялось суммарное время выполнения алгоритмов для соседних лексикографически слов. На основе данных был построен график, изображенный на графике ниже (Рис 7). Реализованный программный код на языке программирования C++ содержится в приложении.

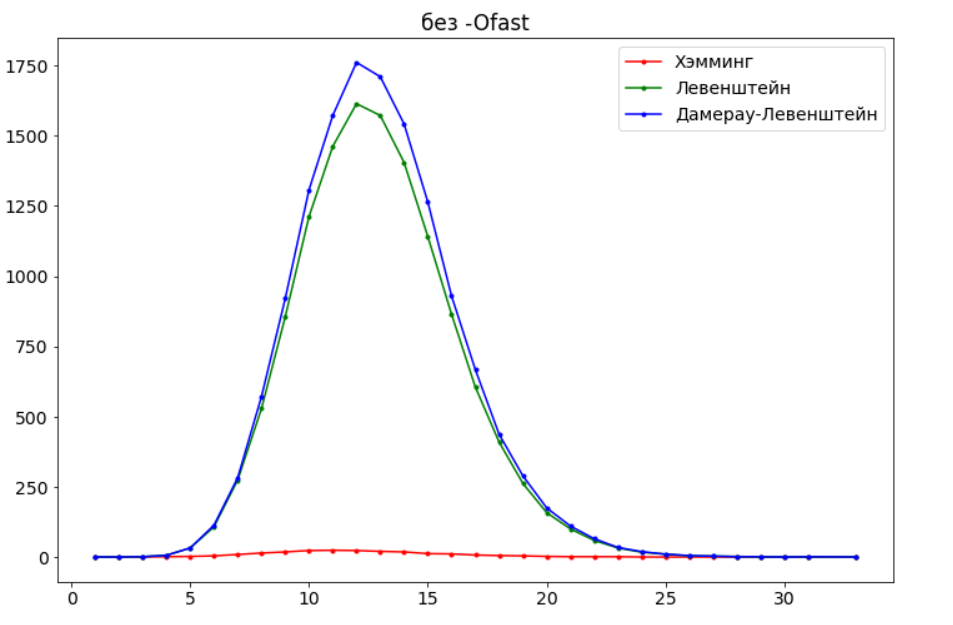


Рис 7 – График времени выполнения алгоритмов без оптимизации Ofast

После тщательного анализа вышеупомянутого программного кода было определено, что оптимизации во время компиляции могут значимо снизить время исполнения программы. Ниже приводится график времён исполнения алгоритмов для соответствующих метрик в зависимости от длин слов с оптимизацией Ofast при компиляции (Рис. 8).

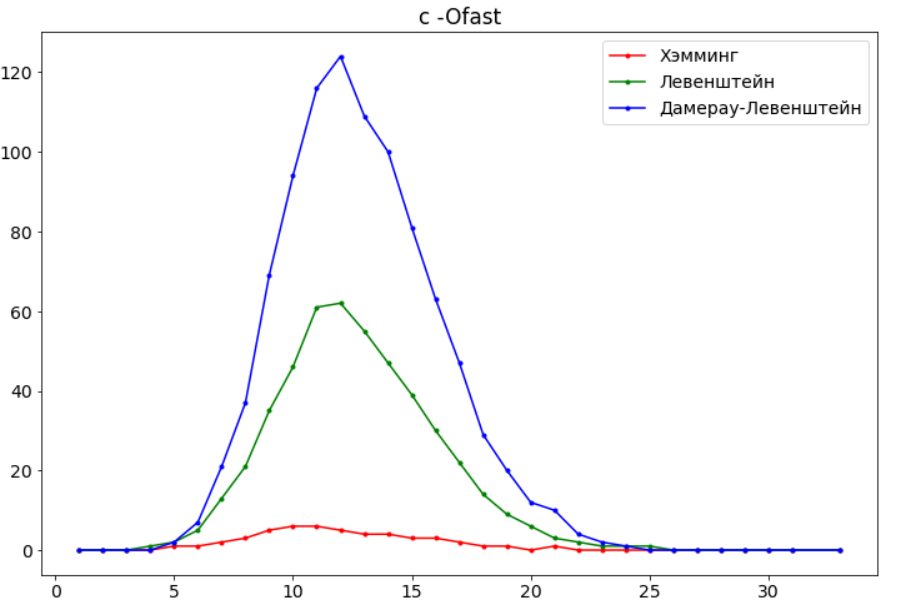


Рис 8 - График времени выполнения алгоритмов с оптимизацией Ofast

Анализируя результаты, можно утверждать, что для программной реализации с оптимизацией на этапе компиляции можно использовать самую оптимальную по точности метрику – метрику Дамерау-Левенштейна, которая позволяет для одной длины слова находить равное ему за 124 миллисекунды, что является достаточно быстрым для требуемой скорости исполнения программы.

2.2 Выбор методологии для проектировки android приложения

Рассматриваемые в первом разделе принципы для построения архитектуры android приложения можно свести с следующим основным концепциям:

1. Приложение состоит из модулей
2. Модули могут взаимодействовать между собой
3. В случае отказа одного из модулей оставшиеся не прекращают работу
4. Ни один из модулей не выполняет никакие сложные вычислительные процессы, а также не оперирует большим объёмом данных

Используя все вышеупомянутые принципы, была спроектирована следующая архитектура (Рис. 9):

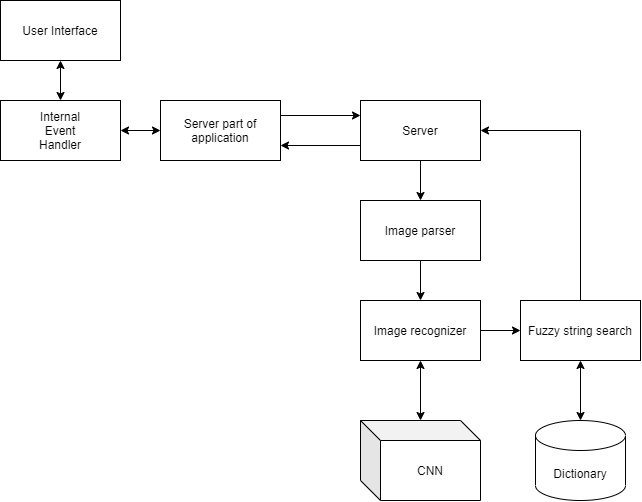


Рис 9 – Архитектура android-приложения и сервера

**2.3 Выводы**

В данном разделе был приведён процесс разработки метода для улучшения качества распознавания русского рукописного текста с использованием нечёткого поиска, а также была разработана архитектура android приложения для использования данного метода.

Раздел 3. Разработка android приложения и метода оптимизации

В данном разделе происходит выбор языка программирования для android приложения, а также рассматривается реализация алгоритма нечёткого поиска и реализация android-приложения.

**3.1 Выбор языка программирования для реализации android приложения**

Для реализации android-приложения требуется выбрать язык программирования и среду разработки. Ниже приводится список самых популярных языков программирования для мобильной разработки под android:

1. Java
2. Kotlin
3. Python
4. JavaScript
5. C++

Рассматривая данный список стоит выделить два самых популярных для разработки под android языка программирования – Java и Kotlin. В 2018 году основной разработчик всего программного обеспечения и самой системы android – компания Google, заявила об изменении выбора основного языка программирования для разработки на android с Java на Kotlin. Однако, в силу инертности рынка, данный язык всё ещё является, основываясь на данных с PYPL[1], в 10 раз менее популярным, нежели язык программирования Java. По этой причине из двух наиболее подходящий для разработки языков программирования под android для создания данного приложения будет взят за основу язык программирования Java.

Классической средой разработки для написания android приложений является android-studio. Данная среда и будет использоваться в процессе разработки.

**3.2 Программная реализация метода оптимизации и android приложения**

TODO

* 1. **Выводы**

В данном разделе был произведён выбор языка программирования для android приложения, а также была рассмотрена реализация алгоритма нечёткого поиска и реализация android-приложения.

**Раздел 4. Экспериментальные исследования android приложения и метода нечёткого поиска**

В данном разделе производится тестирование разработанного метода нечёткого поиска, а также разработанного android приложения, подводятся выводы по итогам разработки.

**4.1 Тестирование разработанного приложения**

**4.2 Выводы по результатам тестирования**

[**Заключение**](#page18)

**Литература**

**Приложение**