Изображение выглядит как Цвет электрик

Автоматически созданное описание

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| НАСИБУЛЛИН РУСЛАН РАДИКОВИЧ  ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ АЛГОРИТМА ПОИСКА ПОДСТРОКИ В СТРОКЕ ХЭШИРОВАНИЕМ ПО СИГНАТУРЕ | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **КУРСОВОЙ ПРОЕКТ** | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| по дисциплине «Структуры и алгоритмы компьютерной обработки данных»  по образовательной программе подготовки бакалавров по направлению  02.03.03 – Математическое обеспечение и администрирование информационных систем, профиль «Технология программирования» | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | |  |  | | Студент гр. Б9122 | | | |  | | | | Насибуллин Р.Р. | | | | |
|  |  | |  | | | | (подпись) | | | |  | | | | |
|  | | | | | | | | | | |  |  | | Руководитель доцент департамента ПИИИ | | | | | | | | | | |  | |
|  | | | | | | | | | | |  |  | | к.т.н., |  | | | | | | С.Н. Остроухова | | | | | |
|  | | | | | |  |  | | | |  |  | |  | (подпись) | | | | | |  | |  | | | |
|  |  |  |  | | | | | |  | |  | |  |  | | «\_\_» | |  | | | | | | 2023 г. | |
|  | | | | | | | | | | | | | | (подпись) | |  |  | |  |  | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Защищен оценкой | | | | | | | |  | | | | |  | | | | | | | | | | | | | |
|  | | | |  |  | | | | | | | |  | | | | | | | | | | | | | |
| (подпись) | | | |  | Фамилия И.О. | | | | | | | |  | | | | | | | | | | | | | |
| «\_\_\_\_» | | |  | | | | | | | 2023 г. | | | | | | | | | | | | | | | | |
|  | | |  | | | | | | |  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2023 | | | | | | | | | | | | | | | | | | | | | | | | | | |

Оглавление

[Введение 2](#_Toc171422473)

[1 Алгоритм хэширования по сигнатуре 5](#_Toc171422474)

[1.1 Основные определения и понятия 5](#_Toc171422475)

[1.2 Описание хэширования по сигнатуре 5](#_Toc171422476)

[1.3 Анализ алгоритма хэширования по сигнатуре 6](#_Toc171422477)

[2 Описание компьютерной программы 8](#_Toc171422478)

[2.1 Требования к генерации 8](#_Toc171422479)

[2.2 Требования к эксперименту 9](#_Toc171422480)

[2.3 Архитектура программного средства 10](#_Toc171422481)

[2.4 Сценарий диалога 12](#_Toc171422482)

[3 Описание эксперимента 17](#_Toc171422483)

[3.1 Эксперименты над быстрой и интроспективной сортировками 17](#_Toc171422484)

[Заключение 19](#_Toc171422485)

[Список литературы и интернет-источников 20](#_Toc171422486)

# Введение

В эпоху информационных технологий обработка и анализ информации занимает одно из самых важных мест в жизни человека. С каждым годом количество информации в виде электронного текста существенно возрастает. Для облегчения работы с большими объемами текстовых данных требуются алгоритмы, помогающие быстро и точно находить нужную информацию. Существует множество алгоритмов четкого поиска, но далеко не всегда задачи по поиску решаются нахождением точного соответствия. Например, необходимо найти слово, точно написание которого мы не знаем. Возникают задачи, требующие поиска по сходству.

В наше время алгоритмы неточного поиска изучаются, так как объёмы текстовой информации растут и, соответственно, повышаются требования к скорости поиска.

На данный момент разработано множество алгоритмов поиска подстроки в строке. Алгоритм хэширования по сигнатуре – один из алгоритмов решающих задачу неточного поиска подстроки в строке.

**Цель:** провести анализ эффективности алгоритма хэширования по сигнатуре в различных тестовых ситуациях.

**Задачи:**

1. Исследование алгоритма хэширования по сигнатуре с помощью литературных источников, анализ особенностей, выявление критериев производительности.
2. Разработка программного продукта, позволяющего провести экспериментальные исследования поиска подстроки в строке хэшированием по сигнатуре. Программный продукт должен включать в себя такие функции, как: генерация данных, визуализация данных и анализ временной сложности алгоритмов.
3. Проведение экспериментов с применением программного продукта для оценки производительности. Измерение временной сложности алгоритма, построение графиков зависимости между входными данными и длительностью поиска.
4. Анализ результатов эксперимента, определение эффективности алгоритма для различных входных данных. Сравнение производительности обоих алгоритмов с точки зрения временной сложности.

# Алгоритм хэширования по сигнатуре

В этой главе будет рассмотрены основные понятия, описание алгоритма хэширования по сигнатуре, а также анализ этого алгоритма.

## Основные определения и понятия

***Алфавит* –** это упорядоченная последовательность символов языка.

***Слово* –** это последовательность символов алфавита.

***Строка текста* –** последовательность слов, разделённых пробелами, которая используется для хранения текстовой информации.

***Сигнатура слова* –** Задан алфавит А и известны вероятности появления различных символов алфавита. На множестве A задана функция f(a), отображающая буквы в числа от 1 до m (|A|>m>0). Сигнатурой sign(w) слова w назовём вектор размерности m, k-ый элемент которого равняется единице, если в слове w есть символ a такой, что f(a)=k, и в противном случае нулю.

***Хэш* –** битовая строка определённого размера.

***Хэш-функция* –** функция, преобразующая массив входных данных произвольного размера в выходную битовую строку определённого размера в соответствии с определённым алгоритмом.

***Номер сигнатуры* –** значение хэш функции , отображающая множество слов в отрезок целых чисел от 0 до 2m-1.

***Хэш-таблица* –** структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию удаления и операцию поиска пары по ключу.

## Описание хэширования по сигнатуре

Алгоритм хэширования по сигнатуре был описан в статье Бойцова Л.М. “Использование хэширования по сигнатуре для поиска по сходству”. Принцип работы алгоритма основан на построении сигнатур слов, используемых в качестве хэша в хэш-таблице.

Общая идея алгоритма состоит в следующем:

1. Вычислить хэши для каждого слова в строке и занести их в таблицу
2. Вычислить хэш поискового запроса
3. Перебрать все хэши, отличающиеся от исходного не более чем в k битах

Рассмотрим пример построения хэша. Пусть алфавит A = {а,б,в,г,д,е,ж,з,и,й,к,л,м,н,о,п,р,с,т,у,ф,х,ц,ч,ш,щ,ъ,ы,ь,э,ю,я} – символы кириллицы нижнего регистра. Функция f(a) =2p mod l, где p – порядковый номер символа a в алфавите, l – длина сигнатуры. Хэш слова w будем вычислять следующим образом: для каждого символа слова w вычислим f(a) и логически сложим. Таким образом для слова “алгоритм” при длине сигнатуры l=16 хэш будет равен 22797. Пример построения хэша показан на рисунке 1.

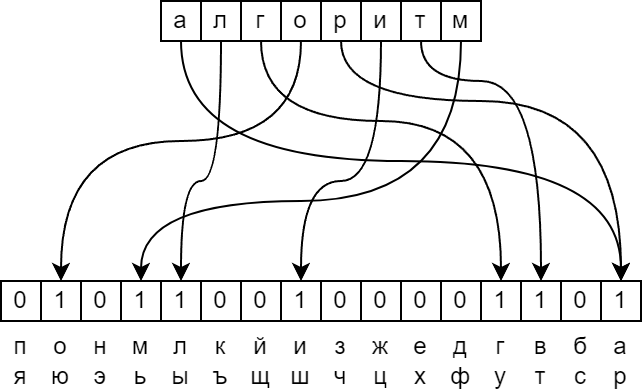


Рисунок 1. Пример построения хэша

## Анализ алгоритма хэширования по сигнатуре

Алгоритм хэширования по сигнатуре обладает весьма привлекательными особенностями: позволяет с высокой скоростью осуществлять поиск на точное равенство и поиск, допускающий одну или две ошибки в поисковом запросе благодаря тому, что основан на хэш таблице. среднее время работы при k ошибках, длине хэша H для n слов составляет O(|H|k\*n/2|H|).

Его недостатком является то, что он требует много памяти и времени для создания словаря. Также при малом размере хэша сильно страдает точность поиска. Из того, что при замене одного символа могут изменяться сразу два бита, алгоритм, реализующий искажения не более 2 битов одновременно, не выдаст полного объема результатов из-за отсутствия значительной (в зависимости от отношения размера хэша к алфавиту) части слов с двумя заменами. Также этот алгоритм не позволяет проводить префиксный поиск.

# Описание компьютерной программы

В данной главе будет рассмотрена компьютерная программа для проведения экспериментов хэширование по сигнатуре. Компьютерная программа решает такие задачи, как генерация текстовых файлов с возможностью указать длину генерируемого текста, слов, а также алфавит, из которого будут генерироваться слова, применение вышеупомянутого алгоритма нечеткого поиска в ранее сгенерированных файлах, фиксирование времени его работы и построение гистограмм, демонстрирующих влияние различных параметров на производительность работы алгоритма поиска.

## Требования к генерации

1. Требования к входным данным.
   * 1. *N* – количество слов в тексте*,*

*N* ∈ [1 .. 10000000]

* + 1. *S1* – минимальная длина слова,

*S1* ∈ [1 .. 32]

* + 1. *S2* – максимальная длина слова,

*S2* ∈ [1 .. 32]

* + 1. *A* – алфавит

*A* ⊂ {а,б,в,г,д,е,ж,з,и,й,к,л,м,н,о,п,р,с,т,у,ф,х,ц,ш,щ,ъ,ы,ь,э,ю,я} либо *A* пустое

1. Требования к выходным данным.

F – файл, удовлетворяющий следующим критериям:

* Файл имеет расширение txt;
* В файле должны быть записаны последовательности символов кириллицы нижнего регистра. Элементы записаны в одну строку, слова разделяются пробелом.

1. Функциональные требования

* КП должна позволять:

а) Вводить количество генерируемых слов;

б) Вводить минимальный и максимальный размер генерируемых слов;

в) Указывать символы алфавита, из которых генерируются слова;

* КП должна создавать файл и генерировать его содержимое в соответствии с заданными входными данными;
* КП должна размещать созданный файл в директории проекта;
* КП должна считать количество сравнений, проведенных каждой из сортировок в процессе работы.
* КП должна считать количество слов, найденных алгоритмом нечеткого поиска.
* КП должна засекать время работы алгоритма нечеткого поиска.
* КП должна строить диаграммы на основании множества выходных данных.
* КП должна при создании нового файла позволить ввести название файла или выбрать существующий для перезаписи.
* КП должна проверять корректность введенных данных:
  + КП должна проверять, чтобы введенное пользователем *A* было подмножеством требуемого множества символов;
* КП должна сообщать пользователю об ошибке в случае некорректности введенных данных;

## Требования к эксперименту

1. Входные данные

FN – множество имен файлов F, удовлетворяющих критериям, указанным в пункте 2.1 – B.

1. Выходные данные
2. FN – Множество файлов удовлетворяющие следующим условиям:
   * Каждый FN файл имеет расширение txt;
   * Каждый n-ый файл содержит строку состоящую из N слов состоящих из символов алфавита   
     *A* ⊂ {а,б,в,г,д,е,ж,з,и,й,к,л,м,н,о,п,р,с,т,у,ф,х,ц,ш,щ,ъ,ы,ь,э,ю,я}
3. Гистограммы, отображающие:

* зависимость времени работы алгоритма нечеткога поиска
* зависимость количество найденных слов от длины хэша

1. Функциональные требования
   * + КП должна позволять выбирать файлы для эксперимента в директории компьютера;
     + КП должна засекать время работы нечеткого поиска хэшированием по сигнатуре и считать количество найденных слов, а затем по полученным данным строить соответствующие гистограммы.

## Архитектура программного средства

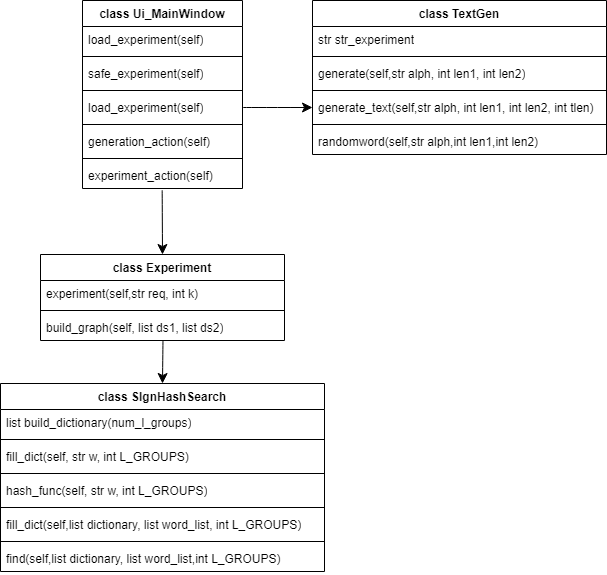


Рисунок 3. Архитектурный проект

Архитектурный проект, реализующий эксперимент поиском хэшированием по сигнатуре, представлен на рисунке 3. Для реализации программы используются следующие классы:

1. *Ui\_MainWindow* – класс, наследник от класса экранной формы, реализующий главное окно программы, через которое генерируются текстовые файлы, а так же обрабатываются события пользовательских действий (нажатия кнопок, выбор элементов и т.д.)
2. *Generation* – класс, реализующий методы генерации различных текстов. Методы этого класса, вызываемые из методов класса *Ui\_MainWindow*, проверяют корректность данных и на основе переданных параметров (характеристик последовательности) производят генерацию. Имеет следующие публичные методы:
   * *randomword* – создает строку, размера от S1 до *S2* содержащую символы алфавита A.
   * *generate* – создает строку, размера от S1 до *S2* содержащую символы алфавита A и добавляет её к str\_experiment.
   * *generate\_text* – создает строку str\_experiment из N слов, размера от S1 до *S2* содержащую символы алфавита A.
3. *Experiment* – класс, реализующий запуск методов класса SignHashSearch, подсчёт времени работы и количества найденных слов, а также построение диаграмм.
4. *SignHashSearch* – класс, реализующий поиск хэшированием по сигнатуре:

* build\_dictionary(int num\_l\_groups) – выделяет память под словарь dictionary, представляющий из себя массив 2num\_l\_groups-1 множеств.
* fill\_dict(self, str w, int L\_GROUPS) – распределяет слова из списка слов word\_list по словарю dictionary согласно значению хэша слова.
* hash\_func(self,str w, int L\_GROUPS) – вычисляет хэш строки w, состоящий из L\_GROUPS бит строки w.
* find(self, list dictionary, int k, str req, int L\_GROUPS) – осуществляет нечеткий поиск запроса req в словаре dictionary с отличием в k бит в хэше.

## Сценарий диалога

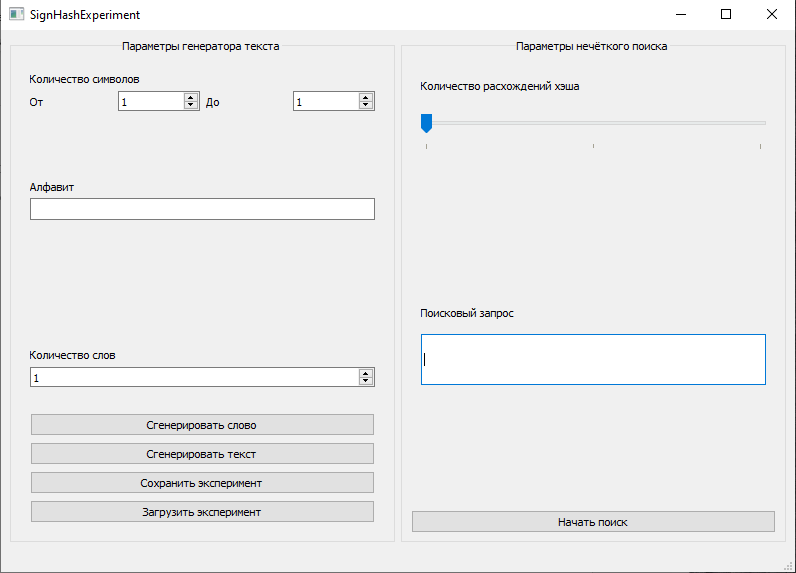
При запуске программы появляется основное окно «SignHashExperiment», представленное на рисунке 4.

Рисунок 4. Главная экранная форма

На данном этапе пользователь может выбрать количество слов генерируемого текста, длину генерируемых слов и алфавит и ввести алфавит из которого будут генерироваться слова, сгенерировать текст, а затем найти поисковый запрос и провести эксперимент. Для генерации текста необходимо нажать на кнопку «Сгенерировать текст».

1. При нажатии кнопки «Сгенерировать текст»:
2. Если поле «Алфавит» пустое, то создаётся строка текста с указанным числом слов, содержащая слова длины указанного диапазона, состоящие из символов кириллицы нижнего регистра.
3. Если поле «Алфавит» непустое, то создаётся строка текста с указанным числом слов, содержащая слова длины указанного диапазона, состоящие из символов поля.
4. При нажатии кнопки «Сгенерировать слово»:
5. Если поле «Алфавит» пустое, то создаётся слово длины указанного диапазона, состоящее из символов кириллицы нижнего регистра и добавляется к текстовой строке.
6. Если поле «Алфавит» непустое, то создаётся слово длины указанного диапазона, состоящее из символов поля и добавляется к текстовой строке.
7. При нажатии кнопки «Сохранить эксперимент»:

Появляется окно выбора файлов, в котором можно выбрать файла расширения .txt, чтобы совершить перезапись, либо ввести название нового текстового файла

1. При нажатии кнопки «Загрузить эксперимент»:

Появляется окно выбора файлов, в котором можно выбрать файл расширения .txt, чтобы считать из него текст.

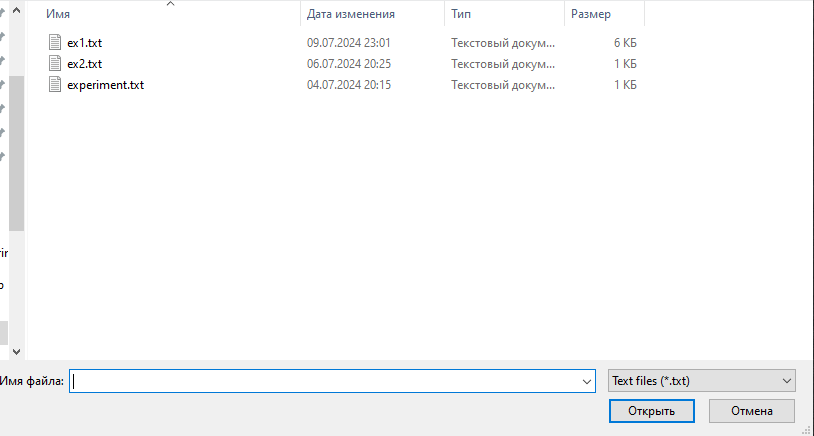


Рисунок 5. Окно выбора файлов для сортировки

1. При нажатии кнопки «Начать поиск»:

Проводится неточный поиск хэшированием по сигнатуре слова в поле «Поисковый запрос» с расхождениями в хэше указанными ползунком «Количество расхождений хэша» и выводятся диаграммы.

Диаграммы отображают следующие сведения по оси OY: время построения словаря – синим цветом, время поиска – зелёным цветом, количество найденных слов – красным цветом, по оси OX: размер хэша. Пример диаграммы представлен на рисунке 6.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 6. Диаграммы работы поиска

# Описание эксперимента

В данной главе будут описаны проведенные над нечётким поиском хэшированием по сигнатуре эксперименты, а также обоснованы их результаты.

## Эксперименты над хэшированием по сигнатуре

1. ***Первый эксперимент.***

Цель эксперимента: показать, что эффективность поиска хэшированием по сигнатуре зависит от количества слов в тексте и длины хэша.

Входные данные: четыре различных текста длиной 1000, 10000, 100000,1000000 слов по 7 символов, 0 отличий в хэше, поисковый запрос – “мочалка”.

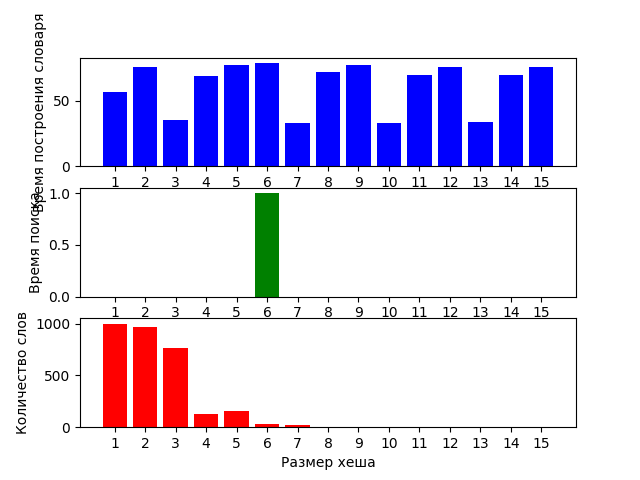
Выходные данные: результат работы поиска представлен на рисунках 12-15. 

Рисунок 12. результат работы на строке из 1000 слов

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 13. Результат работы на строке из 10000 слов

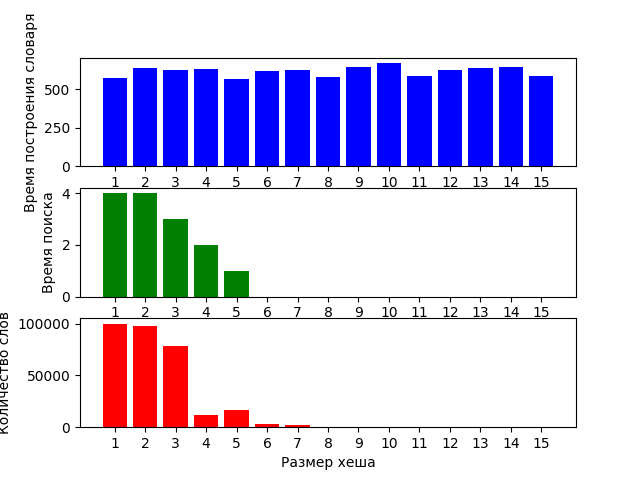


Рисунок 14. Результат работы на строке из 100000 слов

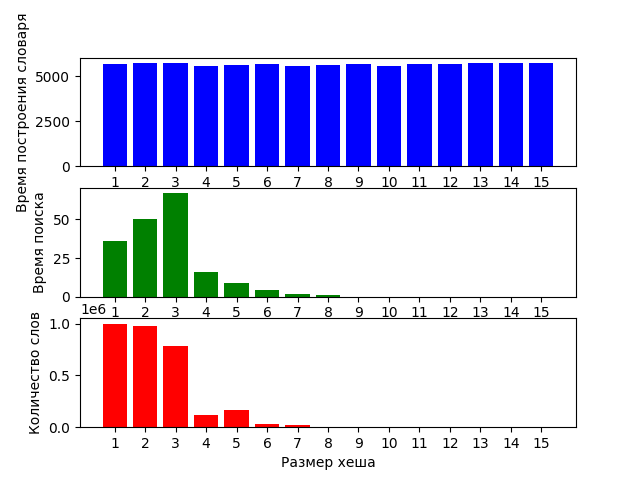


Рисунок 15. Результат работы на строке из 1000000 слов

Обоснование результата: С ростом количества слов растёт время, затрачиваемое на построение словаря. Большое время поиска при малых размерах хэша обусловлено присоединением больших списков найденных слов к выходному списку слов. С увеличением размера хэша, количество слов, хэш которых совпадает с хэшем запроса уменьшается, из-за чего количество элементов в этих списках тоже уменьшается, и длительность поиска становится сравнима с O(1).

1. ***Второй эксперимент.***

Цель эксперимента: показать, что эффективность поиска хэшированием по сигнатуре зависит от количества расхождений в хэше.

Входные данные: текст длиной 10000 слов по 7 символов, поисковый запрос – “мочалка”. Повтор с количествами ошибок в хэше 0, 1 и 2.

Выходные данные: результат работы поиска представлен на рисунке 16.

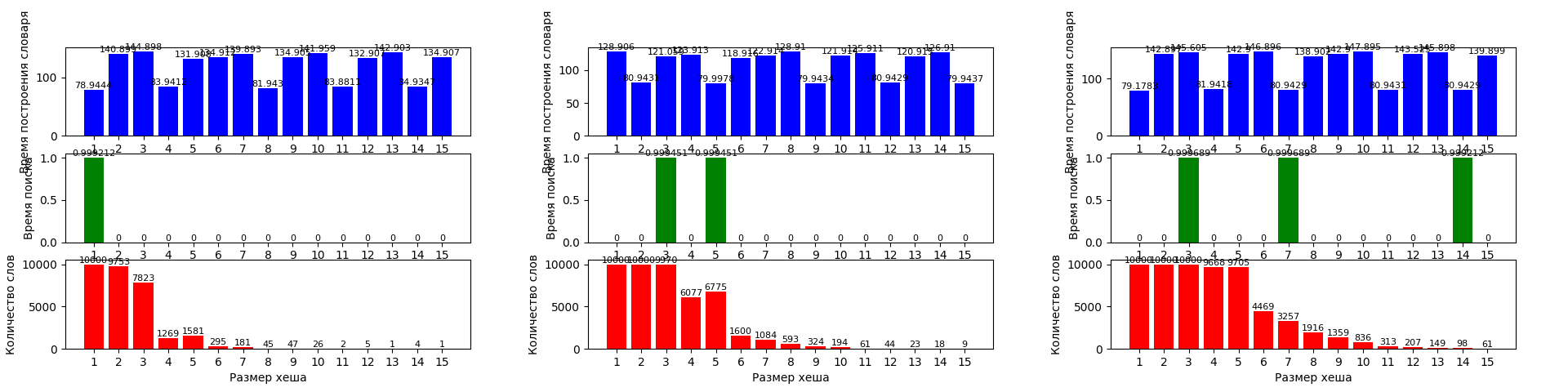


Рисунок 16. Результат работы поиска

Обоснование результата: Время поиска растёт при увеличении количества ошибок в хэше, однако всё равно остаётся малым. Сильно увеличивается выборка слов.

# Заключение

В рамках данного курсового проекта была разработана компьютерная программа, выполняющая генерацию текстовых строк и ставящая эксперименты над различными сгенерированными строками.

На основе проведенных экспериментов можно сделать вывод о том, что эффективность метода сортировки в общем случае зависит от количества слов, длины хэша и количества ошибок в хэше.

Таким образом, при выборе алгоритма сортировки, большое значение имеют размеры сортируемых последовательностей и их тип.

# Список литературы и интернет-источников

1. Сметанин Н. Нечёткий поиск в тексте и словаре [Электронный ресурс] // habr – Коллективный блог – 2011. – Режим доступа: https://habr.com/ru/articles/114997/.
2. Бойцов Л.М. “Использование хэширования по сигнатуре для поиска по сходству” [Электронный ресурс] // ВМК МГУ – Коллективный блог – 2013. – Режим доступа: https://cs.msu.ru/sites/cmc/files/docs/boycov.pdf