# Практическая реализация алгоритмов перебора и анализа в контексте криптографических и лингвистических задач: Опыт оптимизации временной сложности

Работу выполнил ученик 4 курса
группы ИСП-11.1
Асанов Тимофей Андреевич
06 февраля 2025
Руководитель: Иванов Сергей Семёнович

- окно консоли
- описание реализации
- вывод работы

## 1 реализация:

**Цель программы:** расшифровать текст зашифрованный шифром виженера с ключом размера 3 символа путём перебора.

Шифр Виженера — придуман французским дипломатом Блезом де Виженером в 1586 году. В XVII–XVIII веках он был широко популярен благодаря своей стойкости, а в 1863 году был взломан Фридрихом Касиски.

Шифр работает на основе полиалфавитического замещения: каждая буква текста сдвигается на значение, определяемое ключевым словом, длина которого повторяется циклически, маскируя частотные закономерности исходного текста.

В контексте этой работы реализация происходила в формате лабораторной работы где буквы ключа были заменены на их числовые значения напрямую. [1]

**Способ:**

Текст расшифровывается поочерёдно ключами от 000 до 999.
Каждую итерацию вариант текста делится на слова, и каждое слово сверяется со списком русских слов на полтора миллиона слов на совпадение.
Количество совпавших слов сохраняется в формате {<ключ> : <количество совпадений>}
В конце получившийся словарь сортируется по количеству совпадений.
В результате выводятся первые 400 символов у топ 5 вариантов, с возможностью открытия полного текста.
В первой версии такой перебор занимал примерно 20 минут.

Поэтому мы решили переписать код сравнения слов внутри 1 текста на гибрид C++ и Python.
В результате чего перебор стал занимать приблизительно 1,5 минуты.

Потом была реализована многопоточность, с помощью распределения кодов на 6 процессов.
Выполнение задачи стало занимать примерно 32 секунды.

В завершении мы добавили автоматическое гугление выбранного текста, чтобы выполнить бонусное задание лабораторной - определить автора произведения.

**Важно:**

На момент разработки кода мы не изучили хэш таблицы O(n), бинарный поиск O(n·log m), алгоритм Ахо-Корасик O(n + k), и других методов поиска пересечений двух массивов. [2]

Поэтому мы выбрали самую простую реализацию - линейный перебор O(n·m), с добавлением параллельной обработки в дальнейшем.

**Файловая структура:**

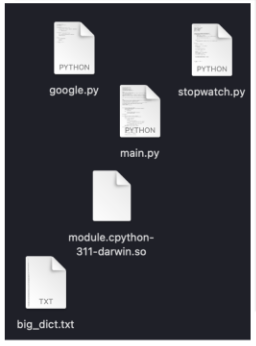
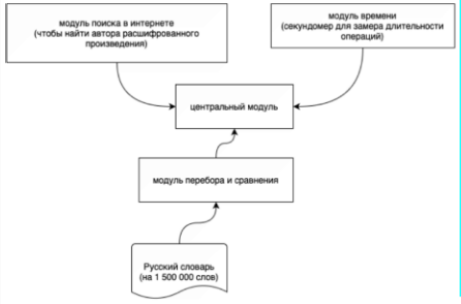



**Схема структуры:**




```
console launch
Last login: Mon Oct 28 20:16:41 on ttys000
вариант(1-15)> 8
формирование и запуск пула процессов
старт процесса перебора 000-166
старт процесса перебора 167-333
старт процесса перебора 334-500
старт процесса перебора 501-667
старт процесса перебора 668-834
старт процесса перебора 835-999
перебор 000-166 окончен
перебор 167-333 окончен
перебор 334-500 окончен
перебор 835-999 окончен
перебор 501-667 окончен
перебор 668-834 окончен
перебор всех вариантов завершён за 45.3977s

BRUTEFORCE RESULT: (отображаются первые 400 символов в первых 5 вариантах)

КОД 137, DICT СОВПАДЕНИЯ 227
милиционер аникин считался самым толстым человеком в деревне. директор маслозавода черкашин весил сто пять килограммов, но участковый уполномоченный был на голову выше его, намного толще, хотя, сколько весит он, никто не знал, так как сам аникин говорил: «а ты попробуй, взвешай меня!» несмотря на полноту, участковый по деревне ходил быстро, особенно в прохладные дни, с людьми поговорить любил, а

КОД 131, DICT СОВПАДЕНИЯ 42
мисицоонлр жничкиу сэитжлсе сжмыт тфлсшщм чесовлкот в децевуе. дицекшор мачлонавфда чецкаюин вечил стф петъ кисогцамтов, нф уэасшкоищй упфлнфмоэенущй быс нж гфлоиу ищил ейо, натнойо шоляе, хошя, скфлъро иесот фн, нирто не знжл, тар кжк чам аноскон йовфрис: «а шц хопцобый, взиешжй тене!» нечмошря на посношу, ущжстровбй хо керлвил хфдис ббстцо, осфбеуно в хрощлакныл дуи, с людами пойовфришъ сюбол, а

* * * * *

выберите текст для полного просмотра (напишите код, например 137)
(0 для выхода)> 137

КОД 137, DICT СОВПАДЕНИЯ 227
милиционер аникин считался самым толстым человеком в деревне. директор маслозавода черкашин весил сто пять килограммов, но участковый уполномоченный был на голову выше его, намного толще, хотя, сколько весит он, никто не знал, так как сам аникин говорил: «а ты попробуй, взвешай меня!» несмотря на полноту, участковый по деревне ходил быстро, особенно в прохладные дни, с людьми поговорить любил, а директора маслозавода черкашина терпеть не мог.

в деревне аникин работал бог знает сколько времени, в каком находился звании, жители не помнили – участковый раз в три года надевал форму, да и то тогда, когда ездил в район. это объяснялось его грандиозной толщиной, и участковый говорил: «если я буду каждый день форму носить, то мне никакой зарплаты не хватит!» летом аникин ходил в широких хлопчатобумажных штанах, в серой рубахе, распахнутой на седой волосатой груди, и в тапочках сорок шестого размера; в грязь он носил кирзовые сапоги, а зимой влезал в серые валенки, от которых его ноги действительно походили на слоновьи.
```

## Вторая реализация:

**Цель программы:** найти все возможные слова состоящие из заданных 5 букв без повторений длиной от 2 до 5.

**Способ:**
1) Генерируется список всех возможных вариантов перестановок букв в слове всех возможных длин
2) Каждый набор букв сверяется со словарём (1,5М слов)
3) Результат сортируется по длине слов и по алфавиту
4) Вывод: Результатов, Времени генерации всех вариантов, Времени анализа всех вариантов, Времени сортировки

**Версии: (некоторые версии получили названия)**
**2-4)** стандартный перебор вариантов и сравнение каждого со словарём

```
введите все буквы подрят без пробелов ››эвокр
кров
вок вор ков кво кор орк окр рок ров эко
вк во ор ок ро
```

**5) (speed update)** для ускорения поиск модифицируется методом нахождения пересечения двух множеств (хэш таблиц)

Была замечена весомая проблема с очисткой/перезаписью переменной после выполнения функции при выполнении поиска слов большей длинны (>10)

**5.5) (middle)** добавлен замер и вывод времени очистки.

```
эрк эск эсп эст эта это
ак ан ап ар ас ат ен ер ес ет ка ке кн ко кр к

Время генерации всех вариантов: 40.4306s
Время анализа всех вариантов: 0.533s
Время сортировки: 0.0005s
очистка...
очистка заняла: 205.1173s
```

**6) (instant dumping) -** безуспешные попытки решить проблему простым перевызовом функции (так как у разных вызовов разная зона памяти)

**7) (INSTANT dumping prosess) -** безуспешные попытки решить проблему выводом вычислений в отдельный процесс с последующим принудительным завершением после получения результатов.

**8)** добавляю вывод более детальных параметров о размере данных и количестве найденных слов.

Выясняется что генерировать множество размером 2 гб оперативной памяти неэффективно и ресурсозатратно.

**9) (hybrid) -** создаём гибридный вариант где только словарь является множеством и выделяется в памяти единоразово.

В нём последовательности букв размеров **меньше 10** обрабатываются подходом поиска пересечения двух множеств,
Т. е. для N слов длины L ≤10:
Сложность = N × O(L²) = N × const = O(N),
Где N — количество коротких слов
учитывая что L² ограничено константой 100.

А последовательности **больше 10** - итеративным подходом (перебором каждого варианта и сравнением со словарём)
Т. е. для N слов длины L >10:
Сложность = O(M × L_max²) - (квадратная зависимость),
Где M — количество длинных слов, L_max — максимальная длина

И **объединённая** сложность такой системы выражается как:
T(N,M) = O(N) + O(M× L_max²) = O(N + M×L_max²)
Где N — количество коротких слов, M — количество длинных слов.
Практическая эффективность зависит от соотношения N и M в конкретных данных.

```
Введите все буквы подрят без пробелов >>реобд

Битовый подход (« <4s)
бедро бердо бодре броде добре
бедо бедр боде бодр боре бред брод добр дроб обде обед обер обре одер одре орде робе роде
бед бер бод бое бор бре бро деб дер доб дор дре дро обе обр оде одр орд оре реб ред роб род рое
бе бо бр де до др ед ер об од ое ор ре ро

Время генерации всех возможных последовательностей: 0.0004s
Кол-во сгенерированных последовательностей: 320
Размер сгенерированных последовательностей: 32.2109kb

Время анализа (поиска слов): 0.0002s
Кол-во найденных слов: 61
Размер найденных слов: 2.2109kb

Время сортировки: 0.0001s
Время очистки: 0.0s


Введите все буквы подрят без пробелов >>ратненокспэ

ИТЕРАТИВНЫЙ ПОДХОД (» >4s)
экспонента
наперсток потрескан простенка экспертно экспонате экспонент экстренна экстренно эскарпное эсперанто
аэротенк коренаст красноте крепостн нантское настроек настроен непроста пакостен покрасне преснота проста
ке протесан протеска растопке скопнена скорпена спаренно странное экранное эксперта экспонат экспорта экс
порте экстерна
екрасно кантоне канторе капроне картоне касторе кнастер констан коренас корнета корсета кранное красное к
расоте кратное крепост кретона кропает ксенона накрест накроет наперто напорет нароете настоек настрое на
троне нектона непарно непрост нестаро окрестн отпарен отпарке отрепан паненок панское пантеон пасторе пат
роне пектора персона покрасн понанес понтера поранен постера потерна проекта прокате пронест просека прос
так простен протеса псарное ракетно расколе распето распнет распоет секатор сектора сенатор скатное скопн

* * * * *

а сте сто стр сэр так тан тар тас тва тек тен теп тер тес тка ткн тко ток тон топ тор тос тпр тра тре тро
тэк эка эко экс энт эра эре эрк эск эсп эст эта это
ак ан ап ар ас ат ен ер ес ет ка ке кн ко кр кс кт на не но нр нс нэ ое ок он оп ор ос от па пе по пр пс
пт ра ре ро рт са се ск сн со сп ср ст та те тк то тр тс тэ эа эк эн эп эр эс эт

Время генерации всех возможных последовательностей: 0s
Кол-во сгенерированных последовательностей: 0
Размер сгенерированных последовательностей: 0.0000kb

Время анализа (поиска слов): 4.6131s
Кол-во найденных слов: 1_371
Размер найденных слов: 128.2109kb

Время сортировки: 0.0005s
Время очистки: 0.0s
```

## Вывод:

Практическая реализация алгоритмов перебора подтвердила их эффективность в решении криптографических и лингвистических задач.

Базовый линейный перебор ключей оказался работоспособным, но требовал оптимизации для сокращения времени выполнения. Переход на гибридный код (C++ и Python) и внедрение многопоточности снизили время расшифровки с 20 минут до 32 секунд. Это демонстрирует ключевую роль технических адаптаций в оптимизации ресурсоёмких операций.

Для лингвистической задачи комбинирование хэш-таблиц и итеративного перебора позволило оптимизировать баланс между скоростью анализа и использованием памяти.

Результаты демонстрируют, что оптимизация временной сложности достигается через комбинацию алгоритмических подходов и инженерных решений.
Полученный опыт может быть применён в разработке инструментов для криптоанализа и автоматической обработки текстовых данных.