*Федеральное государственное бюджетное образовательное учреждение* *высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана (национальный исследовательский институт)»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА ИУ7

**Отчёт**

**по лабораторной работе №8**

**Дисциплина: Анализ алгоритмов**

**Тема лабораторной работы: Эффективный поиск по словарю**

Студент гр. ИУ7-51Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уласик Е.А.** (Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волкова Л.Л.**

(Подпись, дата) (И.О. Фамилия)

Москва, 2019г.

Оглавление

[Введение 3](#_Toc32576999)

[1. Аналитическая часть 4](#_Toc32577000)

[1.1 Эффективный поиск по словарю 4](#_Toc32577001)

[2 Технологическая часть 5](#_Toc32577002)

[2.1 Средства реализации 5](#_Toc32577003)

[2.2 Листинг кода 5](#_Toc32577004)

[3 Экспериментальная часть 6](#_Toc32577005)

[3.1 Анализ результатов 6](#_Toc32577006)

[3.2 Вывод 7](#_Toc32577007)

[Заключение 8](#_Toc32577008)

# [Введение](#__RefHeading___Toc3482_2555955926)

Поиск по словарю является задачей, которая стоит во многих сферах программирования. Поиск по словарю является задачей, которую требуется решать быстро, поэтому необходимы методы для оптимизации данной задачи.

Целью данной работы является изучение способа эффективного поиска по словарю.

Для достижения цели требуется решить следующие задачи.

1. Изучить применение частотного анализа в задаче эффективного поиска по словарю.
2. Реализовать алгоритм эффективного поиска по словарю.
3. Сравнить алгоритмы эффективного и базового поиска по словарю по времени работы.
4. Сделать выводы о применимости алгоритма эффективного поиска к решению задачи поиска по словарю.

# [1. Аналитическая часть](#__RefHeading___Toc3484_2555955926)

В этом разделе будет дано описание алгоритма эффективного поиска по словарю.

## 1.1 Эффективный поиск по словарю

Алгоритм на вход получает словарь. Сначала в словаре необходимо провести частотный анализ. Для этого нужно взять первый элемент каждого значения в словаре по ключу и подсчитать частотную характеристику, т.е. сколько раз этот элемент встречается в качестве первого элемента. Таким образом мы повторяем алгоритм для i-го элемента каждого значения, вычисляя для каждого i-го набора частотную характеристику.

По полученным значениям словарь разбивается на сегменты так, что все элементы с одинаковым первым элементом оказываются в одном сегменте. Сегменты сортируются по значению частотной характеристики так, чтобы элементы с наибольшей частотной характеристикой был самый быстрый доступ.

Далее каждый сегмент сортируется по значению. Это необходимо для реализации бинарного поиска в сегменте. Такой поиск обеспечит производительность .

Таким образом, сначала выбирается нужный сегмент, а затем в нем проводится бинарный поиск нужного слова.

# 2 [Технологическая часть](#__RefHeading___Toc3492_2555955926)

## 2[.1 Средства реализаци](#__RefHeading___Toc3494_2555955926)и

Для реализации был выбран язык Python3, так как он хорошо подходит для работы со словарями и списками, а также знаком мне.

## 2[.2 Листинг кода](#__RefHeading___Toc3496_2555955926)

На Листингах 1-2 представлен код программы.

|  |
| --- |
| Листинг 1. Частотный анализ   1. def make\_analyse(): 2. st = dict() 3. for l in letters: 4. st[l] = 0 5. for d in dictionary: 6. st[d['title'][0].lower()] += 1 7. s = [] 8. for l in st: 9. k = {} 10. k['letter'] = l 11. k['count'] = st[l] 12. s.append(k) 13. return s |
| Листинг 2. Эффективный поиск по словарю   1. def binary\_search(value, a): 2. mid = len(a) // 2 3. low = 0 4. high = len(a) - 1 5. while a[mid]['title'] != value and low <= high: 6. if value > a[mid]['title']: 7. low = mid + 1 8. else: 9. high = mid - 1 10. mid = (low + high) // 2 11. return a[mid] 12. def search(word, st): 13. first = word[0] 14. for s in st: 15. if first == s['letter']: 16. res = binary\_search(word, s['array']) 17. return res |

# 3 [Экспериментальная часть](#__RefHeading___Toc3498_2555955926)

## 3[.1 Анализ результатов](#__RefHeading___Toc3500_2555955926)

Для произведения замеров времени выполнения реализаций алгоритмов будет использована формула , где N – количество замеров, t – время выполнения реализации алгоритма, Tn — время выполнения N замеров. Неоднократное измерение времени необходимо для построения более гладкого графика и получения усредненного значения времени.

Количество замеров будет взято равным 100.

Тестирование будет проведено на рандомных словах, выбранных из словаря и начинающихся на каждую из букв алфавита. Тестирование проведено для двух словарей: словаря из 100000 элементов и словаря из 1000000 элементов.

Результаты тестирования приведены на рисунках 1 и 2.

|  |
| --- |
| Рис 1. Диаграмма работы алгоритмов для словаря из 100000 записей |
| Рис 2. Диаграмма работы алгоритма для словаря из 1 000 000 записей |

Как видно из результатов тестирования, эффективный алгоритм работает значительно быстрее простого перебора в обоих ситуациях. Время поиска для всех букв примерно равно, однако время поиска для полного перебора зависит от положения нужного элемента в словаре.

## 3[.2 Вывод](#__RefHeading___Toc3502_2555955926)

В данном разделе был рассмотрен опыт по замеру времени. Выяснено, что эффективный поиск по словарю значительно увеличивает скорость работы программы.

# [Заключение](#__RefHeading___Toc3504_2555955926)

В результате выполнения данной работы были рассмотрены способы решения задачи поиска по словарю с помощью алгоритма оптимизированного поиска поиска и алгоритма поиска полным перебором ключей. Опыт показал, что рекомендуется использовать метод оптимизированного поиска. Было изучено применение частотного анализа в задаче оптимизированного поиска по словарю. Реализован алгоритм оптимизированного поиска по словарю. Сравнены алгоритмы оптимизированного и базового поиска по словарю по скорости работы. Сделаны выводы о применимости алгоритма оптимизированного поиска к решению задачи поиска по словарю.