



Министерство науки и высшего образования Российской
Федерации
Федеральное государственное бюджетное образовательное
учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ
ТЕХНОЛОГИИ (ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ

О Т Ч Е Т

По лабораторной работе № 7

Название: Поиск в словаре

Дисциплина: Анализ Алгоритмов

Студент

ИУ7-52Б

(Группа)

Н.А. Гарасев

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Л.Л. Волкова

(Подпись, дата)

(И.О. Фамилия)

Москва, 2020

Оглавление

Введение.....	3
1. Аналитическая часть.....	4
1.1.Поиск полным перебором	4
1.2.Двоичный поиск в упорядоченном словаре	4
1.3.Сегментный поиск с частным анализом	4
2. Конструкторская часть	5
2.1.Схемы алгоритмов	5
2.2.Частный анализ данных.....	7
3. Технологическая часть	8
3.1.Реализация алгоритмов.....	8
3.2.Тестирование алгоритмов	9
4. Экспериментальная часть.....	10
4.1.Сравнение алгоритмов по времени работы	10
Заключение	11
Список литературы	12

Введение

Цель лабораторной работы: изучить и применить на практике алгоритмы поиска элемента в массиве словарей по ключу. В данной лабораторной работе рассматривается алгоритм перебора, алгоритм бинарного поиска и сегментный поиск с частным анализом данных.

Поиск — обработка некоторого множества данных с целью выявления подмножества данных, соответствующего критериям поиска.

В ходе выполнения лабораторной работы требуется решить следующие **задачи**.

- 1) Реализовать алгоритм перебора.
- 2) Реализовать алгоритм бинарного поиска.
- 3) Реализовать алгоритм сегментного поиска, основанного на результатах частного анализа.
- 4) Сравнить алгоритмы по затраченным ресурсам.

1. Аналитическая часть

Рассмотрим понятия, с которыми мы столкнемся при выполнении лабораторной работы.

1.1. Поиск полным перебором

Алгоритм заключается в том, что для поиска заданного элемента из множества, происходит непосредственное сравнение каждого элемента этого множества с искомым, до тех пор, пока искомый не найдется или множество закончится.

1.2. Двоичный поиск в упорядоченном словаре

Алгоритм поиска заданного элемента в упорядоченном множестве данных заключается в отслеживании наименьшего и наибольшего индексов элементов массива — **min** и **max**, которые равны первому (нулевому) и последнему индексам заданного множества соответственно. Затем вычисляется индекс элемента - **mid**, а также значение элемента, имеющего этот индекс. Значение **mid**, находится через функцию целочисленного деления пополам между значениями **min** и **max**. Если искомое значение элемента множества меньше значения элемента множества с индексом **mid**, то алгоритм начинает новый поиск в левой половине множества; если оно больше — новый поиск ведется в правой половине множества. Если же искомое значение элемента множества равно элементу с индексом **mid**, то алгоритм возвращает его индекс.

1.3. Сегментный поиск с частным анализом

Алгоритм поиска заданного элемента во множестве заключается в том, что множество делится на подмножества (сегменты). Затем эти сегменты сортируются по частоте обращения (использования). Тем самым мы снижаем

трудоемкость у часто вызываемых элементов. Для данного метода необходима первичная обработка данных, которая требует дополнительных ресурсов.

2. Конструкторская часть

Множества представляют собой массив словарей, который состоит из Id пропуска и имени его владельца. Id является универсальным ключом данного множества. На вход алгоритмы принимают ключ. На выходе алгоритм выдает словарь, соответствующий ключу.

2.1. Схемы алгоритмов

На рис. 1-2 приведены схемы алгоритмов поиска.

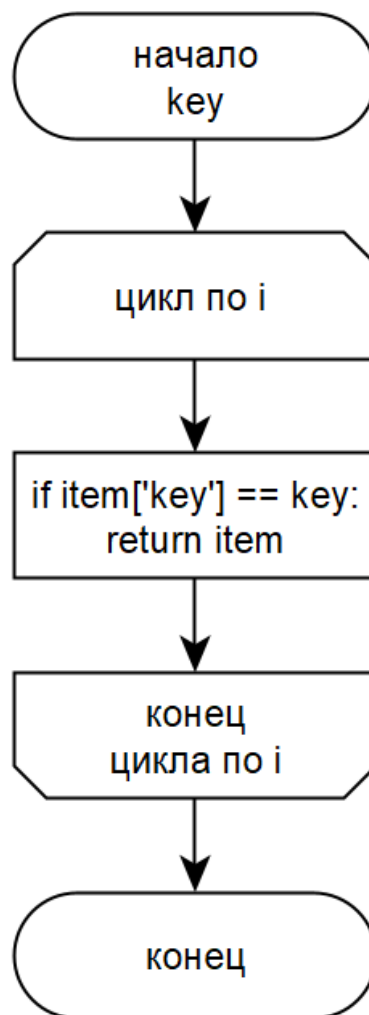


Рисунок 1. Алгоритм полного перебора

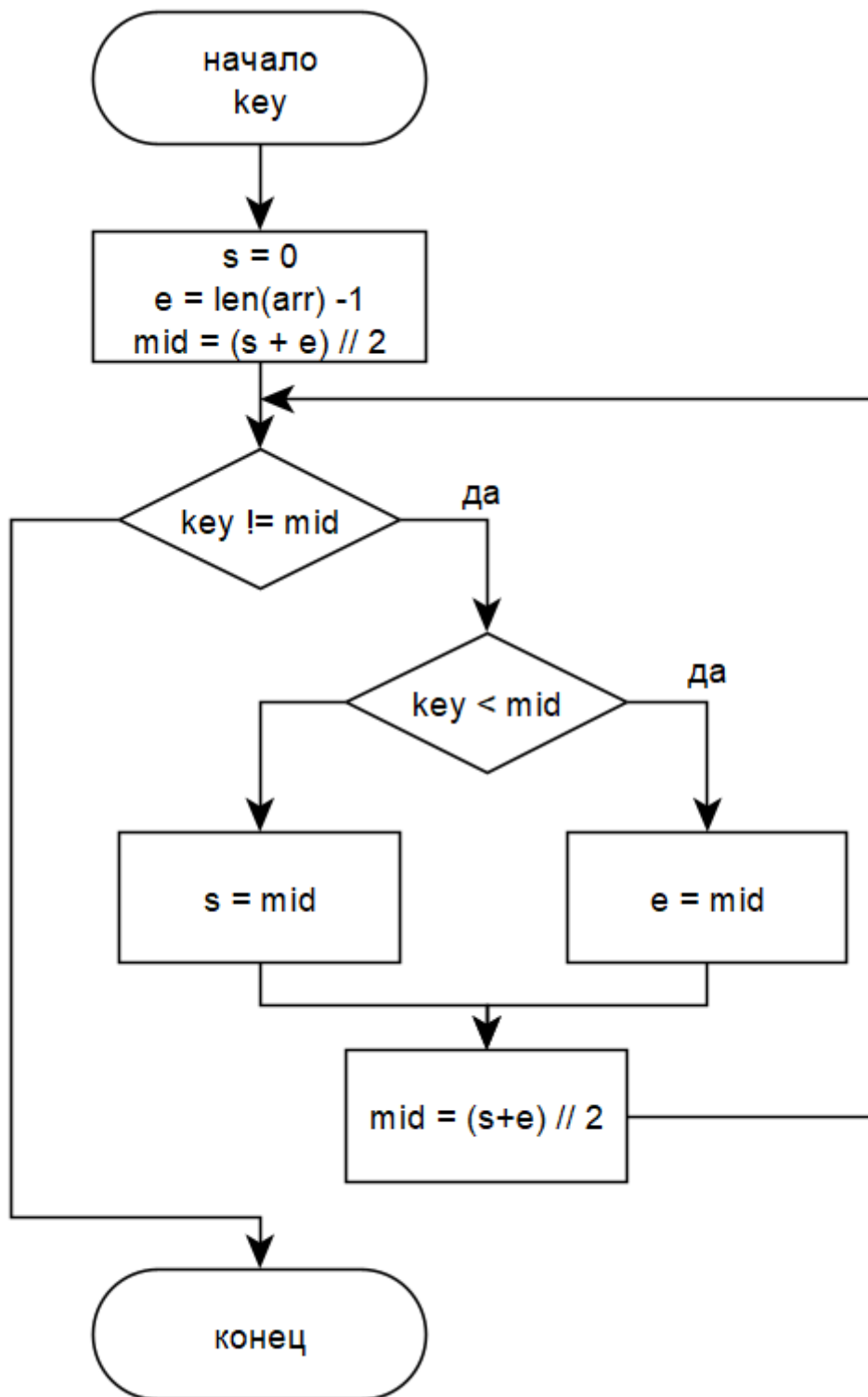


Рисунок 2. Алгоритм бинарного поиска в упорядоченном множестве

Алгоритм сегментного поиска с частным анализом ничем не отличается от алгоритма полного перебора. Разница лишь в том, что это происходит на заранее обработанных данных.

2.2. Частный анализ данных

Данные разбиваются на сегменты, каждому сегменту присваивается своя вероятность. Сегменты с наивысшей вероятностью ставятся в начало массива. При тестировании с тоже вероятностью решается из какого сегмента запрашивать данные.

3. Технологическая часть

В качестве языка программирования был выбран python, т.к. данный язык программирования имеет большое количество полезных библиотек для различных потребностей, а также язык предоставляет средства для быстрого прототипирования и динамической семантики. Для замера процессорного времени была использована функция `process_time()`, стандартной библиотеки python – `time` [2].

3.1. Реализация алгоритмов

В листингах 1-3 представлена реализация алгоритмов сортировки массивов. В листинге 4 представлена функция для замера времени выполнения заданной функции на заданном количестве итераций на матрицах указанного размера.

Листинг 1. Алгоритм полного перебора

```
def brute(self, key):
    for item in self.array:
        if item['key'] == key:
            return item
    return None
```

Листинг 2. Алгоритм бинарного поиска

```
def binary(self, key):
    s = 0
    e = len(self.array) - 1
    mid = (s + e) // 2
    if self.array[s]['key'] > key:
        return None
    elif self.array[e]['key'] < key:
        return None

    if self.array[s]['key'] == key:
        return self.array[s]
    elif self.array[e]['key'] == key:
        return self.array[e]

    tmp = self.array[mid]['key']
    while key != tmp:
        if key < tmp:
            e = mid
        else:
            s = mid
        mid = (s + e) // 2
        tmp = self.array[mid]['key']
    return self.array[mid]
```


Листинг 3. Сегментный алгоритм с частным анализом

```
def segment(self, key):
    if len(self.chance) == 0:
        self.prepare_seg()
    for item in self.array:
        if item['key'] == key:
            return item
    return None
```

Листинг 4. Функция для замера времени выполнения алгоритмов

```
s = Searcher(cnt)
t1 = process_time()
for i in range(repeat):
    a = s.func(i + 100000)
t2 = process_time()
print((t2 - t1) / repeat)
```

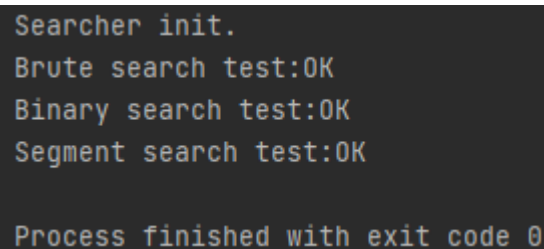
3.2. Тестирование алгоритмов

Для реализованных алгоритмов необходимо провести тестирование. В тестирование входят: граничные случаи, выход за границы и нормальные случаи. Для тестирования алгоритмов создан модуль test.py, листинг которого представлен ниже.

Листинг 5. Функция тестирования алгоритмов

```
err = 0
print('Brute search test:', end='')
for i in range(cnt):
    if s.brute(100000 + i) != s.array[i]:
        err += 1
if err:
    print(err)
else:
    print('OK')
```

Все тесты были пройдены успешно для всех реализованных алгоритмов поиска.



```
Searcher init.
Brute search test:OK
Binary search test:OK
Segment search test:OK

Process finished with exit code 0
```

Рисунок 3. Результат тестирования алгоритмов

4. Экспериментальная часть

Сравним реализованные алгоритмы по времени.

4.1. Сравнение алгоритмов по времени работы

Для получения наиболее точного среднего времени все замеры выполнялись 100 000 раз на словаре длиной 100 000.

Среднее время полного перебора составляет: **2.422e-03** сек.

Среднее время бинарного поиска составляет: **4.844e-06** сек.

Среднее время сегментного поиска (4 сегмента) составляет: **1.847e-03** сек

Среднее время сегментного поиска (5 сегментов) составляет: **1.82e-03** сек

Среднее время сегментного поиска (5 сегментов) составляет: **1.80e-03** сек

Вывод: наглядно видно, что при данных условиях бинарный поиск является самым быстрым алгоритмов поиска. Алгоритм сегментного поиска с частным анализом немного обгоняет алгоритм полного перебора, однако стоит учесть, что дополнительная посегментная сортировка также требует время. Если в программе требуется частое обращение к некоторым частям данных (сегментам), то целесообразно проанализировать их и использовать сегментный поиск. Также можно сказать, что время сегментного поиска зависит от количества сегментов и от вероятности обращения к ним.

Заключение

В ходе работы были изучены и реализованы алгоритмы поиска в массиве словаря по ключу. Был сделан вывод для выбора алгоритма поиска в соответствии с целями использования. Вывод основан на результатах сравнения алгоритмов по среднему времени поиска в массиве по ключу.

Цель работы достигнута. Получены практические навыки реализации алгоритмов поиска, а также проведена исследовательская работа по сравнению этих алгоритмов.

Список литературы

1. Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход. –М.: Техносфера, 2017. – 267 с.
2. Официальный сайт Python, документация [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html>, свободный (дата обращения: 16.09.20).