Лабораторная работа 3 по методам программирования

Создано системой Doxygen 1.9.6

Глава 1

Лабораторная работа номер 3 по курсу "Методы Программирования"

СКБ201 Тур Т.В. Методы Программирования ЛР3

1.1 Введение

Лабораторная работа номер 3 по курсу "Методы программирования". Выполнена студентом Туром Тимофем Владимировичем группы СКБ201.

1.2 Описание

В данной лабораторной работе тредуется реализовать 2 алгоритма хэширования для ключевого поля данных из лабораторной работы 2, построить хэш-таблицы, написать функцию поиска в ней, после чего проверить их работу и сравнить эффективность по времени с предыдущей лабораторной. Мой вариант - 24. Хэширование происходит по полю ФИО.

1.3 Ссылка на репозиторий

Данный проект хранится в репозитории github по ссылке https://github.com/TimothyTur/MP_L3 . В силу явной ненужности многих данных doxygen, они будут отсутствовать там (кроме нужных, например как этот отчет).

1.4 Реализация хэш-таблицы

Эта тема предшествует функицям хэширования, потому что в ней определятся основополагающие параметры. За размерность хэш-таблицы будет взята 2^18, что равно 262144. Этого уже достаточно для 100000 элементов, а предыдущая степень (131072) рискует иметь множество коллизий. Сами коллизии решены методом цепочек.

1.5 "Простая" функция хэширования

Простая хэш-функция реализована через полином по буквам в ключе. За коэффициент полинома взято 31, так как большие буквы будут восприниматься как малые, пробелы будут игнорироваться.

1.6 "Сложная" функция хэширования

Сложная хэш-функция реализована по подобию rot13. В силу ограничения в 2^18 побитовый сдвиг будет взят на 11 и на 7. В теории этот алгоритм быстрее, так как не требует умножения, лишь побитовый сдвиг, также он даст меньше коллизий, так как основан на rot13, в котором в принципе число коллизий минимально.

1.7 Сравнительный график работы поиска

Данный график демонстрирует время, затраченное на поиск элемента. Данные о поисков линейного, multimap, бинарного с сортировкой и просто бинарного взяты напрямую из предыдущей лабораторной.

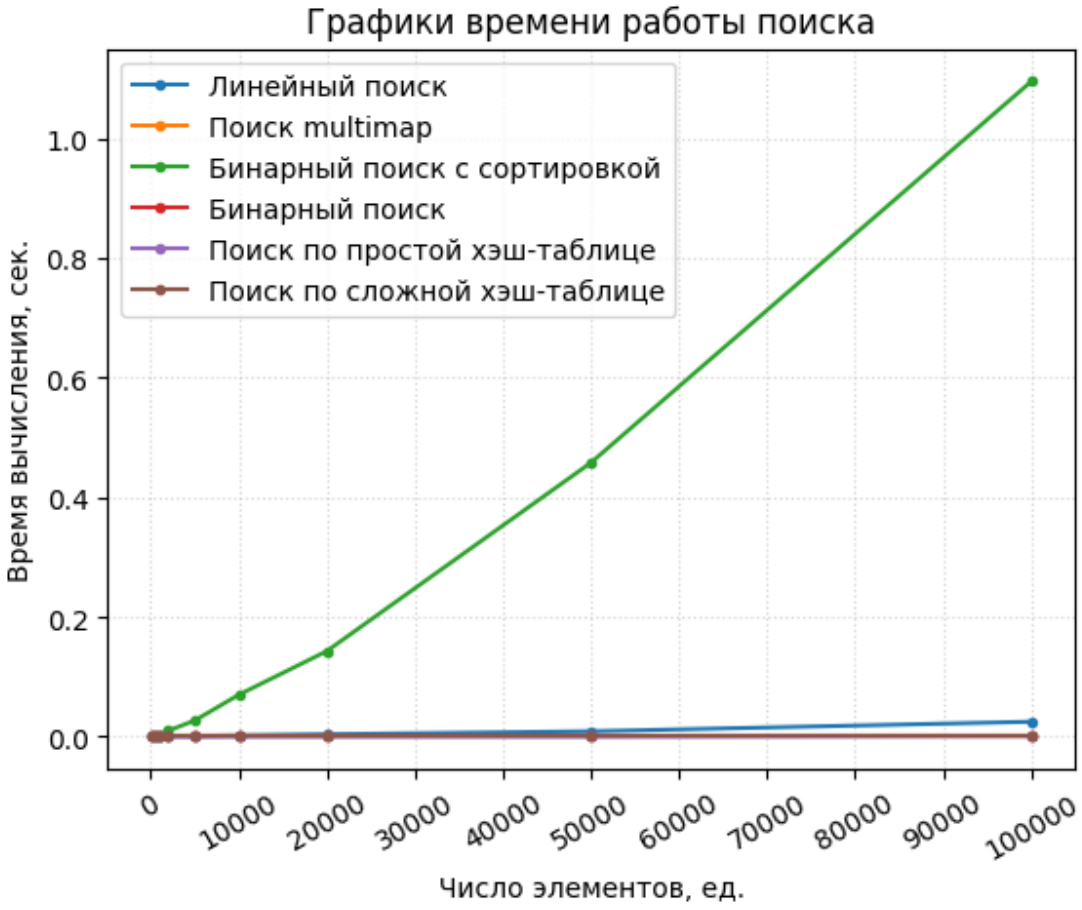

Рис. 1.1 График работы поиска

На данном графике видно, что бинарный поиск, multimap и оба хэша складываются в одну прямую линию в нуле. Бинарному поиску в случае 100 000 элементов требуется не более 17 сравнений.

multimap в случае python реализован на словарях, а они реализованы на хэш-таблицах, а значит его время должно примерно совпадать с новыми измерениями в этой лабораторной. Поиск по хэш-таблице в общем случае требует константное время. Число операций и там и там мало, но видимо достаточно минимально чтобы везде работать почти моментально. Потому имеем прямые и самый быстрый реализованный поиск.

1.8 Сравнительный график числа коллизий

Данный график демонстрирует общее число коллизий в массивах в зависимости от числа элементов в выборке.

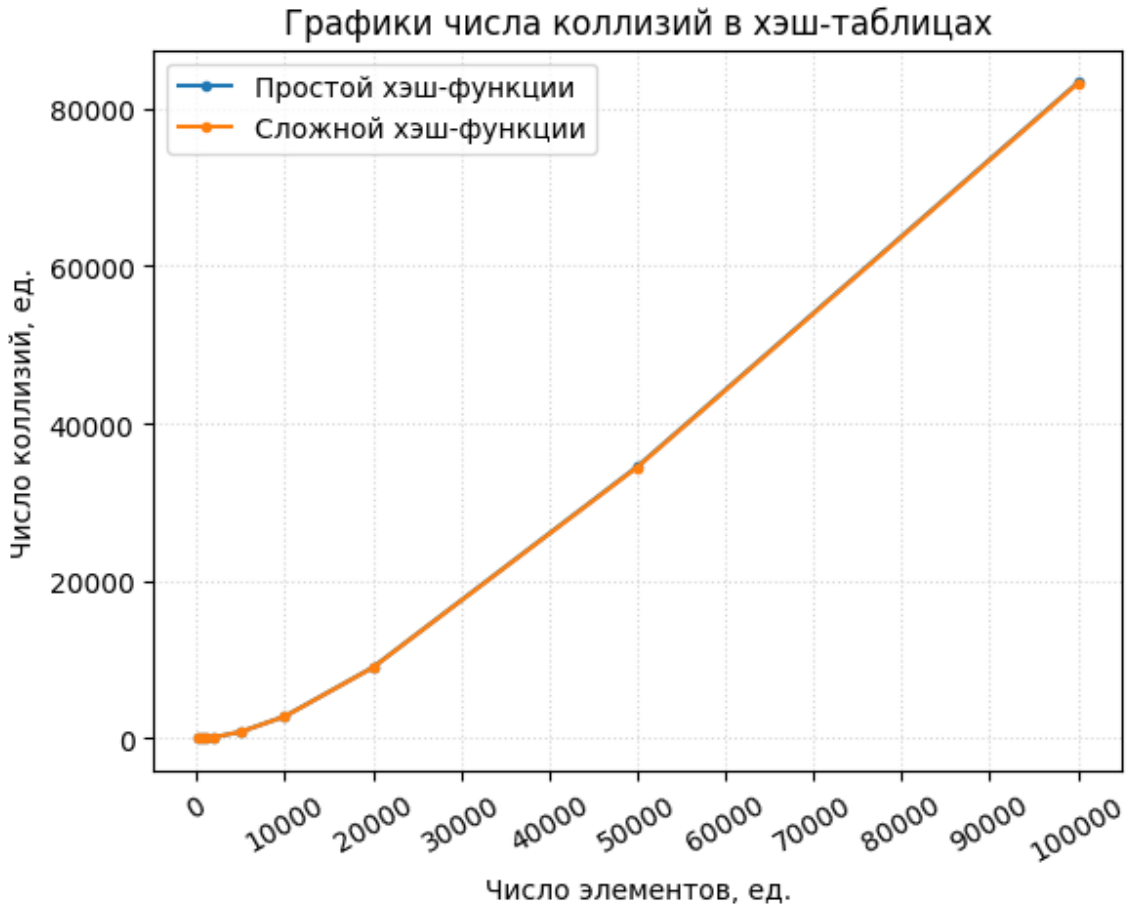

Рис. 1.2 График числа коллизий

На графике особой разницы в числе коллизий не видно. Что странно, так как алгоритмы принципиально разные, и вроде как сложный должен сработать лучше. Но разница видимо настолько никакая, что в виду допустимых погрешностей ее и не видно. Остается только предположить что требуется в разы больший объем выборки. Около миллиона, а то и целого миллиарда. Также велик шанс того что число коллизий обусловленно тем, что в выборке изначально есть элементы с одинаковыми ключами (что продемонстрированно в предыдущей лабораторной).

1.9 Предположение причины числа коллизий

В попытках понять почему получается даже если близкое, но такое огромное число коллизий, я решил ввести счетчики уникальных ключей, используемых в таблице. Тогда каждая заполненная

ячейка вносит в эту переменную вклад в единицу. В то же время функция подсчета коллизий вычисляет количество одинаковых элементов по ключу. На практике это удобно реализовать через возможности связного списка. Посчитав его длину и вычтя 1 получаем число совпадений данного конкретного хэша, а значит сумма по всем заполненным хэшам даст общее число коллизий. Это значит что каждая заполненная ячейка вносит в число коллизий вклад в длину этой ячейки минус 1. Тогда сумма уникальных и сумма коллизий должна совпасть с длинной выборки, так как будем иметь сумму 1 за каждую зуполненную ячейку плюс длина ячейки минус 1. Единицы сокращаются, остается длина ячейки. Сумма длинн по всем ячейкам даст число в принципе распределенных по таблице ячеек, что есть длина исходного распределения. И эта сумма показательна для состоятельной проверки работы программы, так как число уникальных элементов изменяется во время операций над таблицей, а вычисление коллизий - функция, вычисляемая в момент. Эта разница во времени и дает состоятельность, при совпадении чисел. Это было пояснение к тесту, который я сделал, чтобы проверить что все работает правильно. И так и оказалось. Программа работает корректно, но я не совсем понимаю почему тогда столько коллизий. Я перечитал свою же документацию выше, и подумал, а что если совпадение ключей имеет куда большее значение. В первой лабораторной, где генерируется моя выборка, за генератор ФИО я взял учебный список нашей группы, разбил на подэлементы, и генератору буквально сказал выбирать соответственно случайные элементы из полчившегося массива. И в этом и была проблема. В группе нас около 27, что дает 27 имен, 27 фамилий и 27 отчеств (не считая совпадений по группе). Тогда это 27∗27∗27=19683 различных значений. А значит и не удивительно что на выборке длины 100000 имеется целых 80000 коллизий. На чем, получается, можно сделать вывод, что сама выборка ключей изначально не подходит для исследования коллизий в хэштаблице, в виду очень возможных совпадений.

Глава 2

Алфавитный указатель пространств имен

2.1 Package List

Полный список документированных пакетов.

Глава 3

Алфавитный указатель классов

3.1 Классы

Классы с их кратким описанием.

Глава 4

Список файлов

4.1 Файлы

Полный список файлов.

Глава 5

Пространства имен

5.1 Пространство имен full_code

СКБ201 Тур ТВ Методы Программирования ЛР3.

Классы

- class HashTable

  Класс объектов, требуемых по заданию третьей лабораторной работы.
- class MyObject

  Класс объектов, требуемых по заданию первой лабораторной работы.

Переменные

- list ns = [100, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000]
- list t
- int start = -1
- list collisions = [[-1]∗9, [-1]∗9]
- list myLine = [40, 93, 593, 1779, 4102, 4901, 16205, 3801, 55846]
- HashTable table = HashTable()
- None myKey = None
- MyObject elem = MyObject().readOpenedFile(file)
- elemHashBad
- elemOffsetBad
- elemHashGood
- elemOffsetGood
- marker
- rotation
- loc
- True
- alpha
- ls
- bbox_inches

5.1.1 Подробное описание

СКБ201 Тур ТВ Методы Программирования ЛР3.

5.1.2 Переменные

5.1.2.1 alpha

alpha

5.1.2.2 bbox_inches

bbox_inches

5.1.2.3 collisions

list collisions = [[-1]*9, [-1]*9]

5.1.2.4 elem

MyObject elem = MyObject().readOpenedFile(file)

5.1.2.5 elemHashBad

elemHashBad

5.1.2.6 elemHashGood

elemHashGood

5.1.2.7 elemOffsetBad

elemOffsetBad

5.1.2.8 elemOffsetGood

elemOffsetGood

5.1.2.9 loc

loc

5.1.2.10 ls

ls

5.1.2.11 marker

marker

5.1.2.12 myKey

myKey = None

5.1.2.13 myLine

list myLine = [40, 93, 593, 1779, 4102, 4901, 16205, 3801, 55846]

5.1.2.14 ns

list ns = [100, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000]

5.1.2.15 rotation

rotation

5.1.2.16 start

time start = -1

5.1.2.17 t

list t

Инициализатор

```
00001 =  [[0.0, 0.0010027885437011719, 0.0, 0.001001596450805664,
00002      0.0009975433349609375, 0.0019996166229248047, 0.004003763198852539,
00003      0.008996963500976562, 0.025023460388183594],
00004     [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
00005     [0.0, 0.001001596450805664, 0.003000497817993164, 0.009999752044677734,
00006      0.026999473571777344, 0.07000160217285156, 0.14300251007080078,
00007      0.45799732208251953, 1.0950562953948975],
00008     [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
00009     [-1, -1, -1, -1, -1, -1, -1, -1, -1],
00010     [-1, -1, -1, -1, -1, -1, -1, -1, -1]]
```

5.1.2.18 table

HashTable table = HashTable()

5.1.2.19 True

True

Глава 6

Классы

6.1 Класс HashTable

Класс объектов, требуемых по заданию третьей лабораторной работы.

Открытые члены

- def __init__ (self)

  Конструктор хэш-таблицы по заданию лабораторной работы.
- def addBad (self, elem)

  Добавление элемента в хэш-таблицу простого хэша.
- def addGood (self, elem)

  Добавление элемента в хэш-таблицу сложного хэша.
- def getBad (self, addr, step)

  Возвращает элемент таблицы простого хэша
- def getGood (self, addr, step)

  Возвращает элемент таблицы сложного хэша
- def popBad (self, addr, step)

  Удаляет элемент таблицы простого хэша.
- def popGood (self, addr, step)

  Удаляет элемент таблицы сложного хэша.
- def searchBad (self, elem)

  Поиск элемента в таблице простого хэша.
- def searchGood (self, elem)

  Поиск элемента в таблице простого хэша.
- def collisionsBad (self)

  Функция просчитывает текущее число коллизий в таблице простого хэша.
- def collisionsGood (self)

  Функция просчитывает текущее число коллизий в таблице сложного хэша.

Открытые статические члены

- def badHash (obj)

  Вариант простой хэш-функции.
- def goodHash (obj)

  Вариант сложной хэш-функции.

Открытые атрибуты

- bad
- good
- uniBad
- uniGood

6.1.1 Подробное описание

Класс объектов, требуемых по заданию третьей лабораторной работы.

Содержит в себе обе таблицы и весь требуемый функционал для них. В том числе и статические функции вычисления хэша. В таблицах во избежание коллизии сиспользуется метод цепочек. В силу реализации как таковых массивов в python, связный список будет реализован через простой лист.

6.1.2 Конструктор(ы)

6.1.2.1 __init__()

```
def __init__ (
                self )
```

Конструктор хэш-таблицы по заданию лабораторной работы.

Конструктор выделяет массивы для обеих хэш-таблиц.

6.1.3 Методы

6.1.3.1 addBad()

```
def addBad (
                self,
                elem )
```

Добавление элемента в хэш-таблицу простого хэша.

@param elem Элемент для добавления в таблицу

6.1.3.2 addGood()

```
def addGood (
                self,
                elem )
```

Добавление элемента в хэш-таблицу сложного хэша.

@param elem Элемент для добавления в таблицу

6.1.3.3 badHash()

```
def badHash (
                obj )   [static]
```

Вариант простой хэш-функции.

Реализован через полином по буквам в ключе. За коэффициент полинома взято 31.

Аргументы

obj	Объект вычисления хэша. Должен обладать строковым свойством key в русском алфавите.

Возвращает

Вычисленный хэш.

6.1.3.4 collisionsBad()

```
def collisionsBad (
                self )
```

Функция просчитывает текущее число коллизий в таблице простого хэша.

@return Число коллизий.

6.1.3.5 collisionsGood()

```
def collisionsGood (
                self )
```

Функция просчитывает текущее число коллизий в таблице сложного хэша.

@return Число коллизий.

6.1.3.6 getBad()

```
def getBad (
                self,
                addr,
                step )
```

Возвращает элемент таблицы простого хэша

@param addr Хэш искомого элемента.
@param step Сдвиг в цепи элементов

@return Искомый элемент или None, если такого элемента нет

6.1.3.7 getGood()

```
def getGood (
                self,
                addr,
                step )
```

Возвращает элемент таблицы сложного хэша

@param addr Хэш искомого элемента.
@param step Сдвиг в цепи элементов

@return Искомый элемент или None, если такого элемента нет

6.1.3.8 goodHash()

```
def goodHash (
                obj )   [static]
```

Вариант сложной хэш-функции.

Реализова по подобию rot13. В силу ограничения в 2^18 побитовый сдвиг будет взят на 11 и на 7.

Аргументы

obj	Объект вычисления хэша. Должен обладать строковым свойством key в русском алфавите

Возвращает

Вычисленный хэш.

6.1.3.9 popBad()

```
def popBad (
                self,
                addr,
                step )
```

Удаляет элемент таблицы простого хэша.

@param addr Хэш искомого элемента.
@param step Сдвиг в цепи элементов.

@return Возвращает удаленный элемент, None при ошибке.

6.1.3.10 popGood()

```
def popGood (
                self,
                addr,
                step )
```

Удаляет элемент таблицы сложного хэша.

@param addr Хэш искомого элемента.
@param step Сдвиг в цепи элементов.

@return Возвращает удаленный элемент, None при ошибке.

6.1.3.11 searchBad()

```
def searchBad (
                self,
                elem )
```

Поиск элемента в таблице простого хэша.

@param elem Искомый элемент.

@return При успехе, возвращает пару (хэш, смещение), иначе '-1'.

6.1.3.12 searchGood()

```
def searchGood (
                self,
                elem )
```

Поиск элемента в таблице простого хэша.

@param elem Искомый элемент.

@return При успехе, возвращает пару (хэш, смещение), иначе '-1'.

6.1.4 Данные класса

6.1.4.1 bad

bad

6.1.4.2 good

good

6.1.4.3 uniBad

uniBad

6.1.4.4 uniGood

uniGood

Объявления и описания членов класса находятся в файле:

- full_code.py

6.2 Класс MyObject

Класс объектов, требуемых по заданию первой лабораторной работы.

Открытые члены

- def __init__ (self)

  Конструктор класса MyObject Конструктор класса объявляет переменные, которые в нем есть.
- def key (self)

  Выделенное свойство класса - ключ Свойство созданно выделенным, чтобы в разы упросить обращение к нему, подмену для тестов, в то же время не требуя дополнительных ресурсов.
- def __eq__ (self, other)

  Проверка на равенство.
- def __ge__ (self, other)

  Проверка на больше или равно.
- def __gt__ (self, other)

  Проверка на больше.
- def __le__ (self, other)

  Проверка на меньше или равно.
- def __lt__ (self, other)

  Проверка на меньше.
- def __ne__ (self, other)

  Проверка на не равно.
- def __str__ (self)

  Выводит содержимое класса в строке через пробел.
- def writeOpenedFile (self, file)

  Функция записи образа объекта в открытый файл.
- def readOpenedFile (self, file)

  Функция чтения образа объекта с открытого файла.
- def equal (self, other)

  Проверка на точное равенство.

Открытые атрибуты

- badHash
- key
- fio
- num
- din
- dou
- pay
- goodHash

6.2.1 Подробное описание

Класс объектов, требуемых по заданию первой лабораторной работы.

В предыдущей лабораторной был убран генератор класса как таковой, так как он полностью считывается с файла. Поэтому вводить вычисление хэша в конструктор не требуется. Однако это актуально для задачи чтения.

6.2.2 Конструктор(ы)

6.2.2.1 __init__()

```
def __init__ (
                self )
```

Конструктор класса MyObject Конструктор класса объявляет переменные, которые в нем есть.

Не имеет параметров.

6.2.3 Методы

6.2.3.1 __eq__()

```
def __eq__ (
                self,
                other )
```

Проверка на равенство.

@param other Объект сравнения класса MyObject.

@return bool.

6.2.3.2 __ge__()

```
def __ge__ (
                self,
                other )
```

Проверка на больше или равно.

@param other Объект сравнения класса MyObject.

@return bool.

6.2.3.3 __gt__()

```
def __gt__ (
                self,
                other )
```

Проверка на больше.

@param other Объект сравнения класса MyObject.

@return bool.

6.2.3.4 __le__()

```
def __le__ (
                self,
                other )
```

Проверка на меньше или равно.

@param other Объект сравнения класса MyObject.

@return bool.

6.2.3.5 __lt__()

```
def __lt__ (
                self,
                other )
```

Проверка на меньше.

@param other Объект сравнения класса MyObject.

@return bool.

6.2.3.6 __ne__()

```
def __ne__ (
                self,
                other )
```

Проверка на не равно.

@param other Объект сравнения класса MyObject.

@return bool.

6.2.3.7 __str__()

```
def __str__ (
                self )
```

Выводит содержимое класса в строке через пробел.

@return fio, num, din, dou, pay.

6.2.3.8 equal()

```
def equal (
                self,
                other )
```

Проверка на точное равенство.

Требуется для поиска по хэшу

Аргументы

other	Объект сравнения класса MyObject.

Возвращает

bool.

6.2.3.9 key()

```
def key (
                self )
```

Выделенное свойство класса - ключ Свойство созданно выделенным, чтобы в разы упросить обращение к нему, подмену для тестов, в то же время не требуя дополнительных ресурсов.

6.2.3.10 readOpenedFile()

```
def readOpenedFile (
                self,
                file )
```

Функция чтения образа объекта с открытого файла.

Функция также обновляет значения хэша.

Аргументы

file	открытый файл, откуда будет прочтен образ.

6.2.3.11 writeOpenedFile()

```
def writeOpenedFile (
                self,
                file )
```

Функция записи образа объекта в открытый файл.

@param file открытый файл, куда будет записан образ.

6.2.4 Данные класса

6.2.4.1 badHash

badHash

6.2.4.2 din

din

6.2.4.3 dou

dou

6.2.4.4 fio

fio

6.2.4.5 goodHash

goodHash

6.2.4.6 key

key

6.2.4.7 num

num

6.2.4.8 pay

pay

Объявления и описания членов класса находятся в файле:

- full_code.py

Глава 7

Файлы

7.1 Файл full_code.py

Основной исполняевый файл лабораторной работы

Классы

- class HashTable

  Класс объектов, требуемых по заданию третьей лабораторной работы.
- class MyObject

  Класс объектов, требуемых по заданию первой лабораторной работы.

Пространства имен

- namespace full_code

  СКБ201 Тур ТВ Методы Программирования ЛР3.

Переменные

- list ns = [100, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000]
- list t
- int start = -1
- list collisions = [[-1]*9, [-1]*9]
- list myLine = [40, 93, 593, 1779, 4102, 4901, 16205, 3801, 55846]
- HashTable table = HashTable()
- None myKey = None
- MyObject elem = MyObject().readOpenedFile(file)
- elemHashBad
- elemOffsetBad
- elemHashGood
- elemOffsetGood
- marker
- rotation
- loc
- True
- alpha
- ls
- bbox_inches

7.1.1 Подробное описание

Основной исполняевый файл лабораторной работы

@sectioin description Описание Лабораторная работа в изначальном свое виде выполнялась в оболочке "jupyter notebook" в силу его удобства для таких целей. Этот файл является прямым последовательным копированием ячеек из итогового документа (также прикрепленного в github), по причине того что doxygen на файлы ".ipynb" не работает.

7.1.2 Функциональное отличие

В предыдущих лабораторных работах вычисления производились над всеми выборками сразу. В этой же лабораторной хэш-таблица требует слишком много данных, потому вычисления будут происходить последовательно, совершая нужные измерения, после чего удаляя таблицу, приступая к следующей. Также в этой лабораторной работе требуются данные из предыдущей. Данные, используемые здесь, были напрямую скопированные из выводов, сохраненных как часть документации. Также поиск элементов будет осуществляться на первых найденных элементах из предыдущей работы, также сохраненных как часть документации.

7.1.3 Результаты тестирования

Следующая секция представляет из себя набор вывода программы по тестам. Вывод для всех призводится по формату: <размер выборки> <время поиска в простой таблице> <время поиска в сложной> <ключ искомого элемента> <хэш найденного элемента простого алгоритма> <его сдвиг по цепочке> <найденный элемент>

7.1.3.1 <хэш найденного элемента сложного алгоритма> <его сдвиг по цепочке> <найденный элемент>

<повтор для всех размерностей>

100 0.0 0.0 Недомолкин Елизавета Эдикович 237488 0 Недомолкин Елизавета Эдикович 92 2011/04/06 2012/02/27 91881

7.1.3.2 28767 0 Недомолкин Елизавета Эдикович 92 2011/04/06 2012/02/27 91881

500 0.0 0.0 Ташлыков Григорий Николаевич 83313 0 Ташлыков Григорий Николаевич 92 2004/05/30 2013/03/23 32489

7.1.3.3 95710 0 Ташлыков Григорий Николаевич 92 2004/05/30 2013/03/23 32489

1000 0.0 0.0 Гришаев Андрей Сергеевич 74419 0 Гришаев Андрей Сергеевич 78 2004/03/07 2008/10/15 95617

7.1.3.4 168114 0 Гришаев Андрей Сергеевич 78 2004/03/07 2008/10/15 95617

2000 0.0 0.0 Абдуллабеков Илья Николаевна 239180 0 Абдуллабеков Илья Николаевна 96 2004/05/12 2013/11/10 71681

7.1.3.5 233913 0 Абдуллабеков Илья Николаевна 96 2004/05/12 2013/11/10 71681

5000 0.0 0.0 Самунин Тимофей Эдуардович 37393 0 Самунин Тимофей Эдуардович 54 2009/04/23 2009/09/30 95564

7.1.3.6 112789 0 Самунин Тимофей Эдуардович 54 2009/04/23 2009/09/30 95564

10000 0.0 0.0 Ташлыков Артём Эдуардович 66484 0 Ташлыков Артём Эдуардович 72 2005/08/13 2010/07/07 68301

7.1.3.7 45054 0 Ташлыков Артём Эдуардович 72 2005/08/13 2010/07/07 68301

20000 0.0 0.0 Красов Илья Александровна 231440 2 Красов Илья Александровна 20 2006/01/24 2014/04/03 37206

7.1.3.8 184951 2 Красов Илья Александровна 20 2006/01/24 2014/04/03 37206

50000 0.0 0.0 Осипова Радомир Ашотович 139305 0 Осипова Радомир Ашотович 99 2007/12/03 2008/08/06 21649

7.1.3.9 39101 0 Осипова Радомир Ашотович 99 2007/12/03 2008/08/06 21649

100000 0.0 0.0 Грицун Илья Сергеевич 154063 27 Грицун Илья Сергеевич 42 2009/06/14 2011/04/01 60423

7.1.3.10 247281 27 Грицун Илья Сергеевич 42 2009/06/14 2011/04/01 60423

Предметный указатель