**Правительство Российской Федерации**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Федеральное государственное автономное образовательное учреждение

высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ "ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Кафедра «Компьютерная безопасность»

**ОТЧЕТ**

**К ЛАБОРАТОРНОЙ РАБОТЕ №3**

**По дисциплине**

|  |
| --- |
| Работу выполнил  Студент группы СКБ 192 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.О.Ташкинов  подпись, дата  Работу проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.А. Драчев  подпись, дата  Москва 2022 |

**«Методы программирования»**

**Постановка задачи**

**Лабораторная работа №3**

1) Придумать «простую» (не сильно вдаваясь в вероятность коллизий) и «сложную» (более эффективную по скорости и с меньшим числом коллизий) хэш-функции вычисления хэша ключевого поля своего варианта (своего класса).

2) Добавить поле со значением хэша в класс, изменить конструкторы и

методы соответствующим образом.

3) Построить хэш таблицу на основе значения хэша и написать функцию поиска элемента в массиве объектов с использованием хэш-таблицы, реализовать один из методов разрешения коллизий.

4) Провести эксперименты с исследованием зависимости времени поиска от размерности массива для обоих хэш-функций, построить графики, сделать выводы.

5) Сравнить полученные результаты с результатами времени поиска, полученными в предыдущей работе.

6) Исследовать зависимость числа коллизий для каждой хэш-функции от размерности массива, построить график.

Вариант 14:

Массив данных с расписанием поездов:

Расписание содержит номер поезда, дата отправления, тип поезда, (скорый, пассажирский, товарный), время отправления, время в пути (сравнение по полям – дата отправления, время отправления, номер поезда, время в пути (по убыванию))

Поиск по дате отправления

1. **Алгоритм решения задачи**

Данная лабораторная работа была реализована на языке программирования Python 3.9 Графики были построены с помощью Wolfram Mathematica.

В файле Lab1.py определен класс FlightsArr , его конструктор, перегрузки операторов.

В файле Generator.py реализован генератор данных из класса FlightsArr

В файле Lab3.py реализованы алгоритмы поиска хеша от ключевого поля нашего класса, метод разрешения коллизий, алгоритм поиска с помощью хэша и метод создающий хэш-таблицы.

Hash (object) -> int – простая хэш функция

HardHash (object) -> int – сложная хэш функция

HashTable (massiv) – функция создание хэш таблицы по простому хэшу

HashTableHard (massiv) – функция создание хэш таблицы по сложному хэшу

serchHash(hashdict, serch) – функция поиска по хэшу

**2. Выполнение задания**

2.1. Hash (object)

Функция, реализующая “простой” алгоритм поиска хэша от строки.

2.2. HardHash (object)

Функция, реализующая “сложный” алгоритм поиска хэша от строки.

2.3. HashTable (massiv)

Функции, реализующая создание хэш-таблицы и разрешающую возникающие коллизии для “простой” хэш функции.

2.4. HashTableHard (massiv)

Функции, реализующая создание хэш-таблицы и разрешающую возникающие коллизии для “простой” хэш функции.

2.5 serchHash(hashdict, serch)

Функция реализующая поиск строки по значению ее хэша.

2.5. Класс FlightsArr

Данный класс содержит в себе информацию о расписании прибывающих авиарейсах. В класс реализован конструктор, в классе перегружены все операторы сравнения.

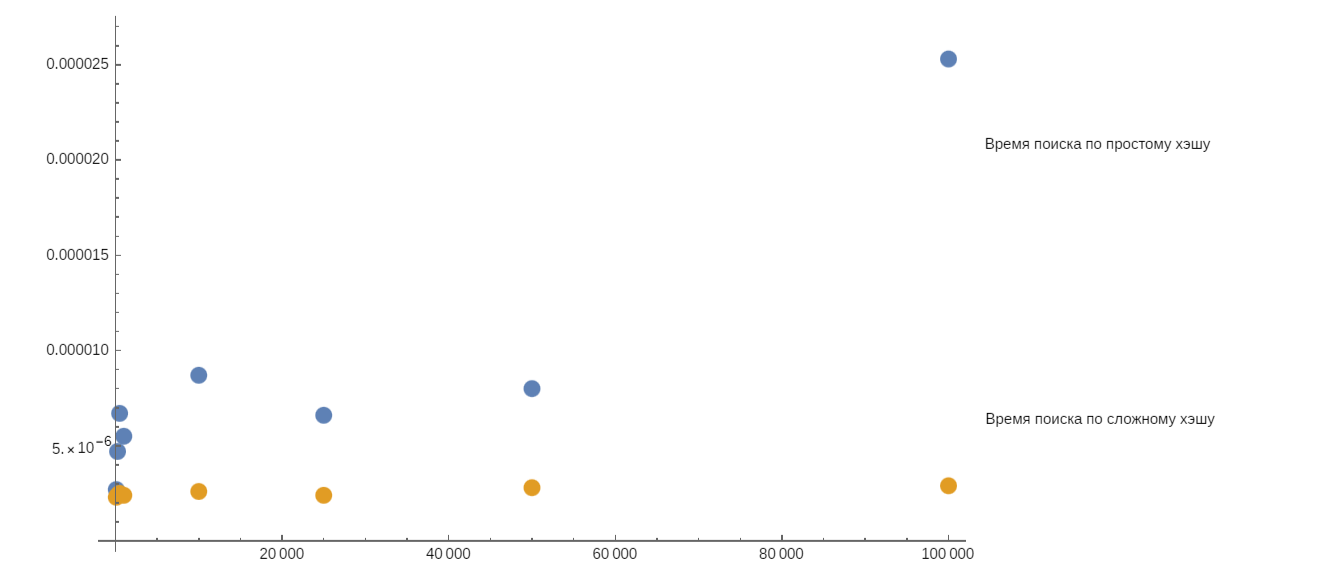
2.7. Функция gen (count: int)

Генерирует случайный массив объектов класса FlightsArr. Принимает в качестве аргумента целое число, равное количеству объектов, которые нужно сгенерировать.

## Построение графиков и выводы

В результате работы программы были получены следующие данные

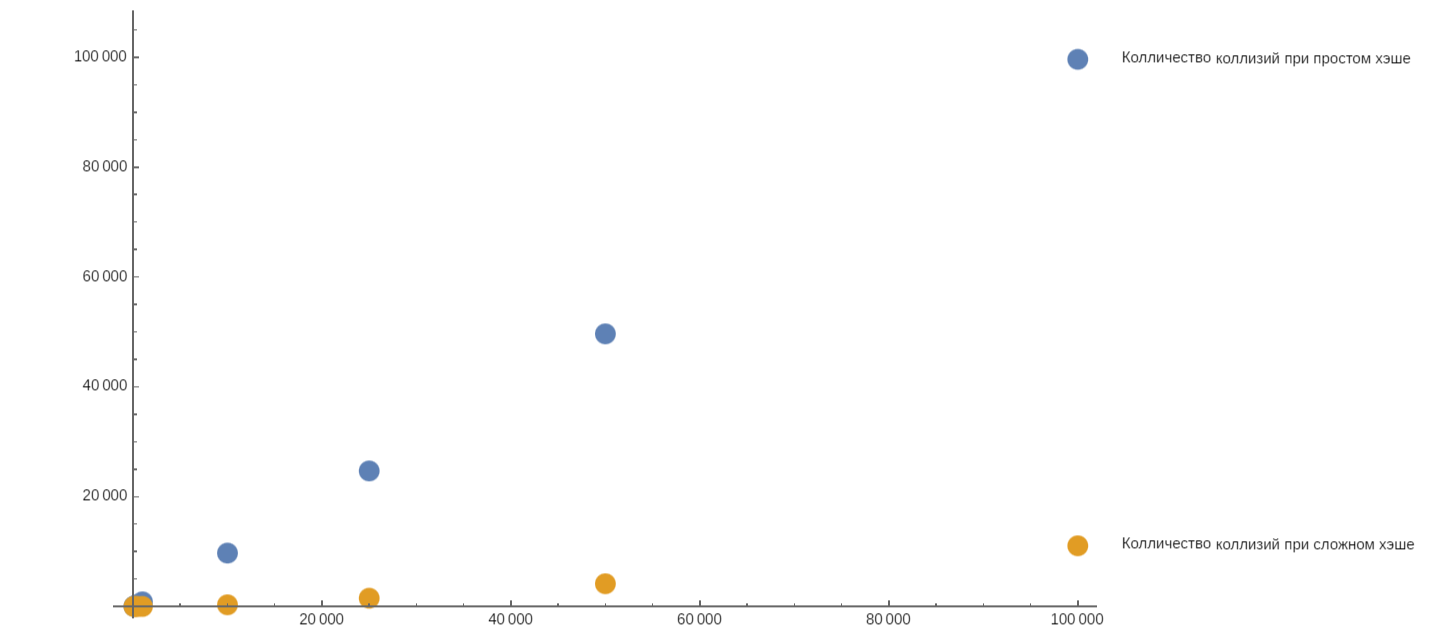
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество элементов | Время линейного поиска, с | Время поиска по словарю, с | Время  бинарного поиска с сортировкой, с | Время бинарного поиска без сортировки, с | Время поиска по простому хэшу, с | Время поиска по сложному хэшу, с |
| 100 | 1.1600\*10^(-5) | 3.9000\*10^(-6) | 0.0005 | 2.7999\*10^(-6) | 2.7000\*10^(-6) | 2.2999\*10^(-6) |
| 250 | 1.5899\*10^(-5) | 2.0000\*10^(-6) | 0.0018 | 4.0000\*10^(-6) | 4.6999\*10^(-6) | 2.3999\*10^(-6) |
| 500 | 2.7899\*10(-5) | 1.6999\*10^(-6) | 0.0045 | 5.6000\*10^(-6) | 6.6999\*10^(-6) | 2.5000\*10^(-6) |
| 1000 | 7.4299\*10(-5) | 2.9000\*10^(-6) | 0.0089 | 6.1000\*10^(-6) | 5.4999\*10^(-6) | 2.3999\*10^(-6) |
| 10000 | 0.0007 | 3.2600\*10^(-5) | 0.1566 | 1.1199\*10^(-5) | 8.6999\*10^(-6) | 2.6000\*10^(-6) |
| 25000 | 0.0022 | 6.3300\*10^(-5) | 0.4010 | 1.3900\*10^(-5) | 6.6000\*10^(-6) | 2.3999\*10^(-6) |
| 50000 | 0.0045 | 9.8299\*10^(-5) | 0.8809 | 1.6400\*10^(-5) | 8.0000\*10^(-6) | 2.7999\*10^(-6) |
| 1000000 | 0.0089 | 0.0001 | 1.9656 | 1.7700\*10^(-5) | 2.5299\*10^(-5) | 2.9000\*10^(-6) |



Из полученных данных можно сделать вывод, что поиск по хэшу эффективнее большинства алгоритмов поиск и сравним по скорости с алгоритмом бинарного поиска в отсортированном массиве. Также можно сделать вывод, что чем меньше коллизий, тем меньше время поиска в хэш таблице, если коллизий вообще не будет, то время поиска почти приблизиться к константе.

Количество коллизий в зависимости от функции хэша

|  |  |  |
| --- | --- | --- |
| Количество элементов | Количество коллизий при работе простой функции хэширования | Количество коллизий при работе сложной функции хэширования |
| 100 | 44 | 0 |
| 250 | 154 | 0 |
| 500 | 371 | 3 |
| 1000 | 845 | 4 |
| 10000 | 9723 | 307 |
| 25000 | 24679 | 1505 |
| 50000 | 49644 | 4143 |
| 100000 | 99628 | 11053 |



**Приложение А**

А.1 исходный код Lab3.py

from Lab1 import FlightsArr

import datetime

import time

from collections import deque

def Hash(object) -> int:

hash = 0

p = 29

p\_pow = 1

str = object

for i in range(len(str)):

hash = (ord(str[i])-ord('a') + 1) + p\_pow

p\_pow \*= p

return hash

def HardHash(object) -> int:

str = object

hash = 0

for i in range(len(str)):

hash = ((hash \* 1664525) + (ord(str[i])-ord('a')) + 1013904223) # ord возвращает код элемента

return hash // 1000000000000000000

def HashTable (massiv):

ct = 0

d = {}

count = len(massiv)

for i in range(count):

if d.get(Hash(massiv[i].name), 0) == 0 :

dq = deque([massiv[i].name])

d.update([(Hash(massiv[i].name), dq)])

else:

spisok = d.get(Hash(massiv[i].name), 0)

if spisok.count(massiv[i].name) != 0:

None

else:

ct += 1

dq = deque()

dq.extend(spisok)

dq.append(massiv[i].name)

d.update([(Hash(massiv[i].name), dq)])

return [d, ct]

def serchHash(hashdict, serch):

hash = Hash(serch)

if hashdict.get(hash, 0) == 0:

return 0

else:

dq = hashdict.get(hash, 0)

return dq.count(serch)

def serchHashHard(hashdict, serch):

hash = HardHash(serch)

if hashdict.get(hash, 0) == 0:

return 0

else:

dq = hashdict.get(hash, 0)

return dq.count(serch)

def HashTableHard (massiv):

d = {}

ct = 0

count = len(massiv)

for i in range(count):

if d.get(HardHash(massiv[i].name), 0) == 0 :

dq = deque([massiv[i].name])

d.update([(HardHash(massiv[i].name), dq)])

else:

spisok = d.get(HardHash(massiv[i].name), 0)

if spisok.count(massiv[i].name) != 0:

None

else:

ct += 1

dq = deque()

dq.extend(spisok)

dq.append(massiv[i].name)

d.update([(HardHash(massiv[i].name), dq)])

return [d, ct]

if \_\_name\_\_ == '\_\_main\_\_':

for n in [100, 250, 500, 750, 1000, 2500, 5000, 7500, 10000, 25000, 50000, 100000]:

arr = []

with open(f"db\_{n}.txt", "r") as db:

data = db.read()

data\_lst = data.split("\n")

for i in data\_lst[:-1]:

per = i.split("--")

per[-5] = datetime.date.fromisoformat(per[-5])

per[-4] = datetime.time.fromisoformat(per[-4])

per[-3] = int(per[-3])

per[-2] = Hash(str(per[-6]))

per[-1] = HardHash(str(per[-6]))

arr.append(FlightsArr(\*per))

print(f"{n}:")

print(f"Колизий просто")

DICT1 = HashTable(arr)

print(DICT1[1])

print(f"Колизий сложно")

DICT2 = HashTableHard(arr)

print(DICT2[1])

t\_start\_1 = time.perf\_counter()

serchHash(DICT1[0], "cannon")

all\_time\_1 = time.perf\_counter() - t\_start\_1

print(f"Поиск просто {all\_time\_1}")

t\_start\_2 = time.perf\_counter()

serchHashHard(DICT2[0], "cannon")

all\_time\_2 = time.perf\_counter() - t\_start\_2

print(f"Поиск сложно {all\_time\_2}")

А.2 исходный код Lab1.py

import datetime

import random

class FlightsArr:

def \_\_init\_\_(self, id, name, date, time, number):

"""

Конструктор класса Flights\_Arr

:param id:

:param name:

:param date:

:param time:

:param number:

"""

self.id = id

self.name = name

self.date = date

self.time = time

self.number = number

def \_\_lt\_\_(self, other):

"""

Перегрузка оператора меньше

:param other: Объект класса :class: `Flights\_Arr`, с которым проводится сравнение

:type other: :class: `Flights\_Arr`

:return: True, если данный объект меньше объекта other, иначе False

:rtype: bool

"""

if self.date < other.date:

return True

elif self.date == other.date:

if self.time < other.time:

return True

elif self.time == other.time:

if self.name < other.name:

return True

elif self.name == other.name:

if self.number > other.number:

return True

return False

def \_\_le\_\_(self, other):

"""

Перегрузка оператора меньше или равно

:param other: Объект класса :class: `Flights\_Arr`, с которым проводится сравнение

:type other: :class: `Flights\_Arr`

:return: True, если данный объект меньше или равен объекта other, иначе False

:rtype: bool

"""

return self.\_\_lt\_\_(other) or (self.date == other.date and

self.time == other.time and

self.name == other.name and

self.number == other.number)

def \_\_gt\_\_(self, other):

"""

Перегрузка оператора больше

:param other: Объект класса :class: `FlightsArr`, с которым проводится сравнение

:type other: :class: `FlightsArr`

:return: True, если данный объект больше объекта other, иначе False

:rtype: bool

"""

return not self.\_\_le\_\_(other)

def \_\_ge\_\_(self, other):

"""

Перегрузка оператора больше или равно

:param other: Объект класса :class: `Flights\_Arr`, с которым проводится сравнение

:type other: :class: `Flights\_Arr`

:return: True, если данный объект больше или равен объекта other, иначе False

:rtype: bool

"""

return not self.\_\_lt\_\_(other)

def \_\_str\_\_(self):

"""

Строковое представление объекта

:return: строковое представление

:rtype: str

"""

return str(self.id) + " " + self.name + " " + str(self.date) + " " + str(self.time) + " " + str(self.number)

def \_\_repr\_\_(self):

return self.\_\_str\_\_()

def bubble\_sort(arr: list) -> list:

"""

Сортировка пузырьком

:param arr: Массив для сортировки

:type arr: list

:return arr: Отсортированный массив

:rtype: list

"""

N = len(arr)-1 # индекс последнего элемента

for i in range(N): # идём от начала до конца массива

for j in range(N): # проходим от начала до конца массива N раз

if(arr[j] > arr[j+1]): # если j элемент массива больше j+1

arr[j], arr[j + 1] = arr[j + 1], arr[j] # меняем значение jго элемена на j+1-ый и наоборот

return arr # возвращаем отсортированный массив

def cocktail\_shaker\_sort(arr: list) -> list:

"""

Шейкер сортировка

:param arr: Массив для сортировки

:type arr: list

:return: Отсортированный массив

:rtype: list

"""

left = 0 # индекс крайнего-левого элемента массива

right = len(arr) - 1 # индекс крайнего-правого элемента массива

while left <= right: # пока индекс первого элемента меньше последнего

for i in range(left, right, +1): # идем от left до right с шагом один

if arr[i] > arr[i + 1]: # если i элемент больше i+1-го

arr[i], arr[i + 1] = arr[i + 1], arr[i] # меняем местами i и i+1-ый элементы

right -= 1 # сдвигаем индекс крайнего-првого элемента влево

for i in range(right, left, -1): # идем от right до left с шагом один

if arr[i - 1] > arr[i]: # если i-1 ый элемент больше i

arr[i], arr[i - 1] = arr[i - 1], arr[i] # меняем местами i и i-1 ый элементы

left += 1 # сдвигаем индекс крайнего-левого элемента вправо

return arr # возвращаем отсортированный массив

def quick\_sort(arr: list) -> list:

"""

Быстрая сортировка

:param arr: Массив для сортировки

:type arr: list

:return: Отсортированный массив

:rtype: list

"""

if len(arr) <= 1: # если длина массива меньше и равна 1

return arr # возвращаем массив

else: # иначе

q = random.choice(arr) # выбираем случайное значение в массиве

l\_arr = [n for n in arr if n < q] # составляем массив из элементов меньше фиксированного

e\_arr = [q] \* arr.count(q) # составляем массив из всех элементов принимащих фиксированное значение

r\_arr = [n for n in arr if n > q] # составляем массив из элементов больше фиксированного

return quick\_sort(l\_arr) + e\_arr + quick\_sort(r\_arr) # рекрсивно выполнеяем программу для левого и правого массива

if \_\_name\_\_ == '\_\_main\_\_':

for n in [100, 250, 500, 750, 1000, 2500, 5000, 7500, 10000, 25000, 50000, 100000]:

arr = []

with open(f"db\_{n}.txt", "r") as db:

data = db.read()

data\_lst = data.split("\n")

for i in data\_lst[:-1]:

per = i.split("--")

per[-3] = datetime.date.fromisoformat(per[-3])

per[-2] = datetime.time.fromisoformat(per[-2])

per[-1] = int(per[-1])

arr.append(FlightsArr(\*per))

print(f"{n}:")

start\_time = datetime.datetime.now()

arr\_bubble = bubble\_sort(arr.copy())

end\_time = datetime.datetime.now()

print(f"Сортировка пузырьком {end\_time - start\_time}")

with open(f"sorted\_bubble{n}.txt", "w") as f:

f.write(str(arr\_bubble))

start\_time = datetime.datetime.now()

arr\_shake = cocktail\_shaker\_sort(arr.copy())

end\_time = datetime.datetime.now()

print(f"Шейкер сортировка {end\_time - start\_time}")

with open(f"sorted\_shake{n}.txt", "w") as f:

f.write(str(arr\_shake))

start\_time = datetime.datetime.now()

arr\_quick = quick\_sort(arr.copy())

end\_time = datetime.datetime.now()

print(f"Быстрая сортировка {end\_time - start\_time}")

with open(f"sorted\_quick{n}.txt", "w") as f:

f.write(str(arr\_quick))

А.3 исходный код Generator.py

from faker import Faker

from Lab1 import FlightsArr

import random

faker = Faker()

Company = {"Компания": ["AiroFlot", "Pobeda", "TurkichAirlains", "NordVind", "S7", "JoPanAirlains", "AmogusAir"]}

def gen(n):

"""

Генерирует n записей вида id-Компания-Дата\_прилета-Время\_прилета-Кол\_во\_пассажиров

:param n: количество генерируемых записей

:type n: int

:return lst: список сгенерированных записей

:rtype lst: list

"""

lst = []

for \_ in range(n):

lst.append(str(str(random.randint(0, 1000)) + "--" + random.choice(Company["Компания"]) + "--" +

str(faker.date\_this\_month()) + "--" + str(faker.time()) + "--" + str(random.randint(1, 300))))

return lst

if \_\_name\_\_ == "\_\_main\_\_":

selections\_lst = [100, 250, 500, 750, 1000, 2500, 5000, 7500, 10000, 25000, 50000, 100000]

for j in selections\_lst:

with open(f"db\_{j}.txt", "w") as db:

for i in gen(j):

db.write(i + "\n")