Национальный исследовательский университет «Высшая школа экономики» Специальность «Компьютерная безопасность»

Отчет
По лабораторной работе №3
«Сортировки»

по дисциплине «Методы программирования»
направления «Компьютерная безопасность»

Вариант 16

Шаплавский Л.П.
СКБ-182

Москва, 2021

Практическое задание по теме «Алгоритмы хэширования».

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..

main.py

```
import data_modul
from StudentClass2 import Hash

findword = "Шварев"
sampl = [100, 500, 1000, 5000, 10000, 50000, 100000]

for j in range(len(sampl)):
    data = []
    asd = Hash()
    data_modul.creation_of_data(sampl[j])
    data_array = data_modul.creation_array2(sampl[j])

    for i in range(len(data_array)):
        asd.insert_of_data(data_array[i])

    print('\n' + '-----' + str(sampl[j]) +  '-----')

    print(asd.value(findword))
```

data_modul.py

```
import random
import StudentClass
import StudentClass2
import datetime

def creation_of_data(N):
    name = ['Шварев','Римский', 'Чайковский', 'Мусоргский', 'Бородин', 'Глинка',
'Скрябин','Рахманинов',
'Стравинский','Прокофьев','Шостакович','Шаплавский','Шимановский','Смирнов','Иванов','Кузнец
ов','Соколов','Попов','Лебедев','Козлов','Новиков','Морозов','Петров','Волков','Соловьёв','В
асильев','Зайцев','Павлов','Семёнов','Голубев']
    fak = ['Факультет экономических наук','Московский институт электроники и математики
им.А.Н. Тихонова','Факультет компьютерных наук','Высшая школа бизнеса','Факультет
права','Высшая школа юриспруденции и администрирования','Факультет
гуманитарныхнаук','Факультет социальных наук']
    spec = [['Департамент теоретической экономики','Департамент
прикладнойэкономики','Департамент математики'],['Департамент электронной
инженерии','Департамент компьютернойинженерии','Департамент прикладной
математики'],['Департамент программной инженерии','Департамент анализа данных
иискусственного интеллекта','Департамент больших данных и информационного
поиска'],['Департамент бизнес-информатики','Департамент
маркетинга','Департаментоперационного менеджмента и логистики'],['Департамент теории права и
межотраслевых юридических дисциплин','Департаментпубличного права','Департамент частного
права'],['Институт юридического менеджмента','Институт кадровогоадминистрирования','Институт
спортивного менеджмента и права'],['Школа философии и культурологии','Школа филологических
наук','Институтклассического Востока и античности'],['Департамент социологии','Департамент
политики и управления','Департаментпсихологии']]
    f1=open('datd.txt','w')#создание файла с данными

    b = random.randint(0, N - 1)

    for i in range(N):
        fak_num = random.randint(0, len(fak) - 1)
        if i == b:
            f1.write('{}'.format(random.randint(100, 300)) + '|' + name[0] + '|' +
fak[fak_num] + '|' + spec[fak_num][random.randint(0, 2)] + '\n')
        else:
            f1.write('{}'.format(random.randint(100,300))+ '|' +
name[random.randint(1,len(name)-1)] + '|' + fak[fak_num] + '|'
+spec[fak_num][random.randint(0,2)] + '\n')
    f1.close()

    return 0

def creation_array(N): #передача данных файла в массив
    f1 = open('datd.txt','r')
    data_array = []
    for i in range(N):
        line = f1.readline()
        data_array.append(StudentClass.Student(line))
    f1.close()
    return data_array

def creation_array2(N): #передача данных файла в массив
    f1 = open('datd.txt','r')
    data_array = []
    for i in range(N):
        line = f1.readline()
        data_array.append(StudentClass2.StudenL(line))
    f1.close()
    return data_array
```

```
def array_info(array): #вывод массива
    for i in range(len(array)):
        array[i].PrintInfo()
    return 0

def output_file(array,Time): #запись данных в файл
    f1 = open('data_output.txt','w')
    for i in range(len(array)):
        f1.write(array[i].PrintInfoSTR())
        f1.write("time:"+str(Time))
        f1.close()
    return 0

def write_data_info(word,time,sampl):
    f1=open('datd_info.txt','a')
    f1.write(str(word)+ '|' + str(time) + '|' + str(sampl) + '|' +
str(datetime.datetime.now()) + '\n'+'\n')
    f1.close()
    return 0

def write_data_info_live(word,time,sampl):
    print(str(word) + '|' + str(time) + '|' + str(sampl) + '|' +
str(datetime.datetime.now()) + '\n' +'_________')
    return 0
```

StudentClass2.py

```
import time


def easy_hash(key, size):
    hashnumber = 0
    for letter in key:
        hashnumber += ord(letter)
    return hashnumber % size


def hard_hash(key, size):
    hashnumber = 0
    for letter in key:
        hashnumber += ord(letter)
        hashnumber -= (hashnumber << 13) | (hashnumber >> 19)
    return hashnumber % size


class StudenL:
    level: int
    name: str
    fac: str
    spec: str
    my_hash_easy: int
    my_hash_hard: int

    def __init__(self, line: str):
        level, name, fac, spec = line.split('|')
        level = int(level)
        self.level = level
        self.name = name
        self.fac = fac
        self.spec = spec
        self.my_hash_easy = easy_hash(name, 1000000)
        self.my_hash_hard = hard_hash(name, 1000000)


class Hash :
    def __init__(self):
        self.size = 1000000
        self.count_easy = 0
        self.count_hard = 0
        self.key_easy = [None] * self.size
        self.data_easy = [[]] * self.size
        self.key_hard = [None] * self.size
        self.data_hard = [[]] * self.size

    def put(self, key, data_, hash_value, slots, data_array, count):
        if slots[hash_value] is None:
            slots[hash_value] = key
            data_array[hash_value] = [data_]
        else:
            if slots[hash_value] == key:
                data_array[hash_value].append(data_)
            else:
                next_slot, count = self.rehash(hash_value, len(slots), count)
                while slots[next_slot] is not None and slots[next_slot] != key:
                    next_slot, count = self.rehash(next_slot, len(slots), count)

                if slots[next_slot] is None:
                    slots[next_slot] = key
                    data_array[next_slot] = [data_]
```

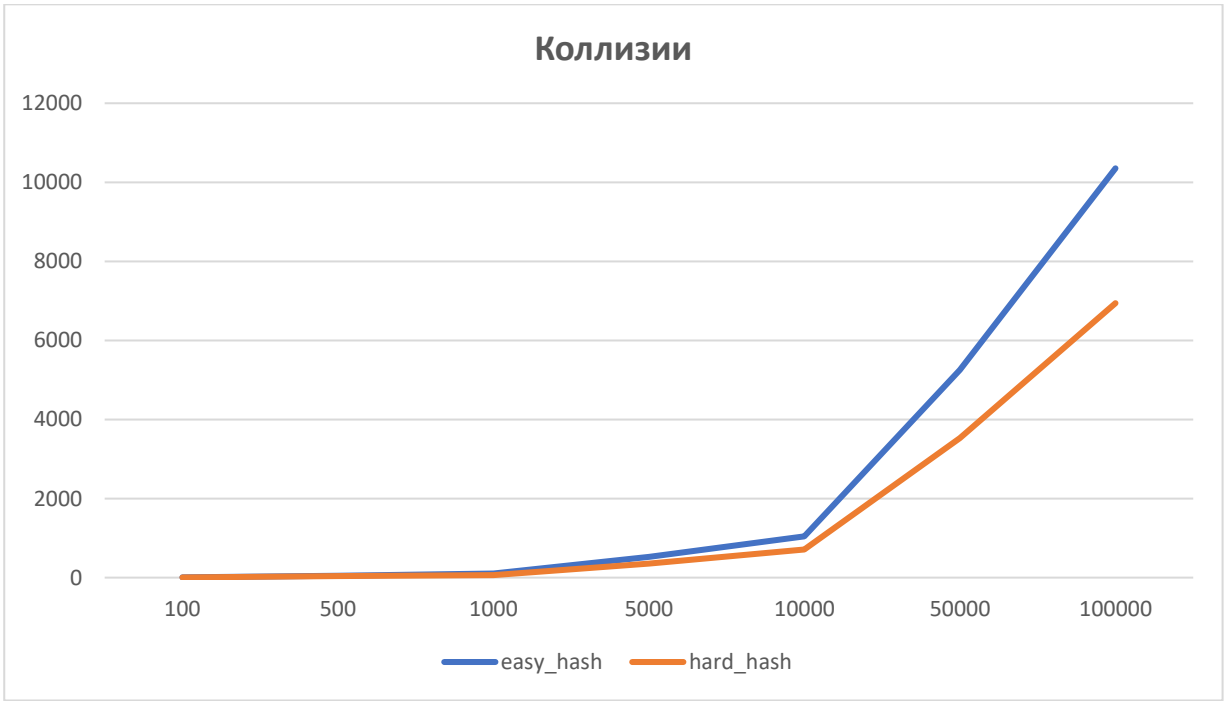
```
                elif slots[next_slot] == key:
                    slots[next_slot] = key
                    data_array[next_slot].append([data_])
                else:
                    raise Exception('Мало места в таблице')
        return count

    def rehash(self, old_hash, size, count=None):
        if count is not None:
            count += 1
            return (old_hash + 1) % size, count
        return (old_hash + 1) % size

    def insert_of_data(self, obj: StudenL):
        self.count_easy = self.put(obj.name, obj, obj.my_hash_easy, self.key_easy,
self.data_easy, self.count_easy)
        self.count_hard = self.put(obj.name, obj, obj.my_hash_hard, self.key_hard,
self.data_hard, self.count_hard)

    def get(self, key, start_slot, slots, data):
        data_ = None
        stop = False
        found = False
        position = start_slot
        while slots[position] is not None and not found and not stop:
            if slots[position] == key:
                found = True
                data_ = data[position]
            else:
                position = self.rehash(position, len(slots))
                if position == start_slot:
                    stop = True
        return data_

    def value(self, key):
        hash_easy_ = easy_hash(key, self.size)
        start_easy = time.time()
        easy = self.get(key, hash_easy_, self.key_easy, self.data_easy)
        time.sleep(0.001)
        end_easy = time.time()

        hash_hard_ = hard_hash(key, self.size)
        start_hard = time.time()
        hard = self.get(key, hash_hard_, self.key_hard, self.data_hard)
        time.sleep(0.001)
        end_hard = time.time()

        if easy:
            return  easy[0].name, str(end_easy - start_easy).replace('.', ','),
self.count_easy, hard[0].name, str(end_hard - start_hard).replace('.', ','), self.count_hard
        else:
            return "Ключ не найден", str(end_easy - start_easy), str(end_hard - start_hard)
```

Результат:

	100	500	1000	5000	10000	50000	100000
easy_hash	0,010001898	0,001001358	0,004311323	0,0044384	0,006973267	0,003666878	0,005284786
hard_hash	0,002997866	0,004321089	0,005717745	0,005308857	0,005628338	0,00537466	0,005219927

Коллизии:

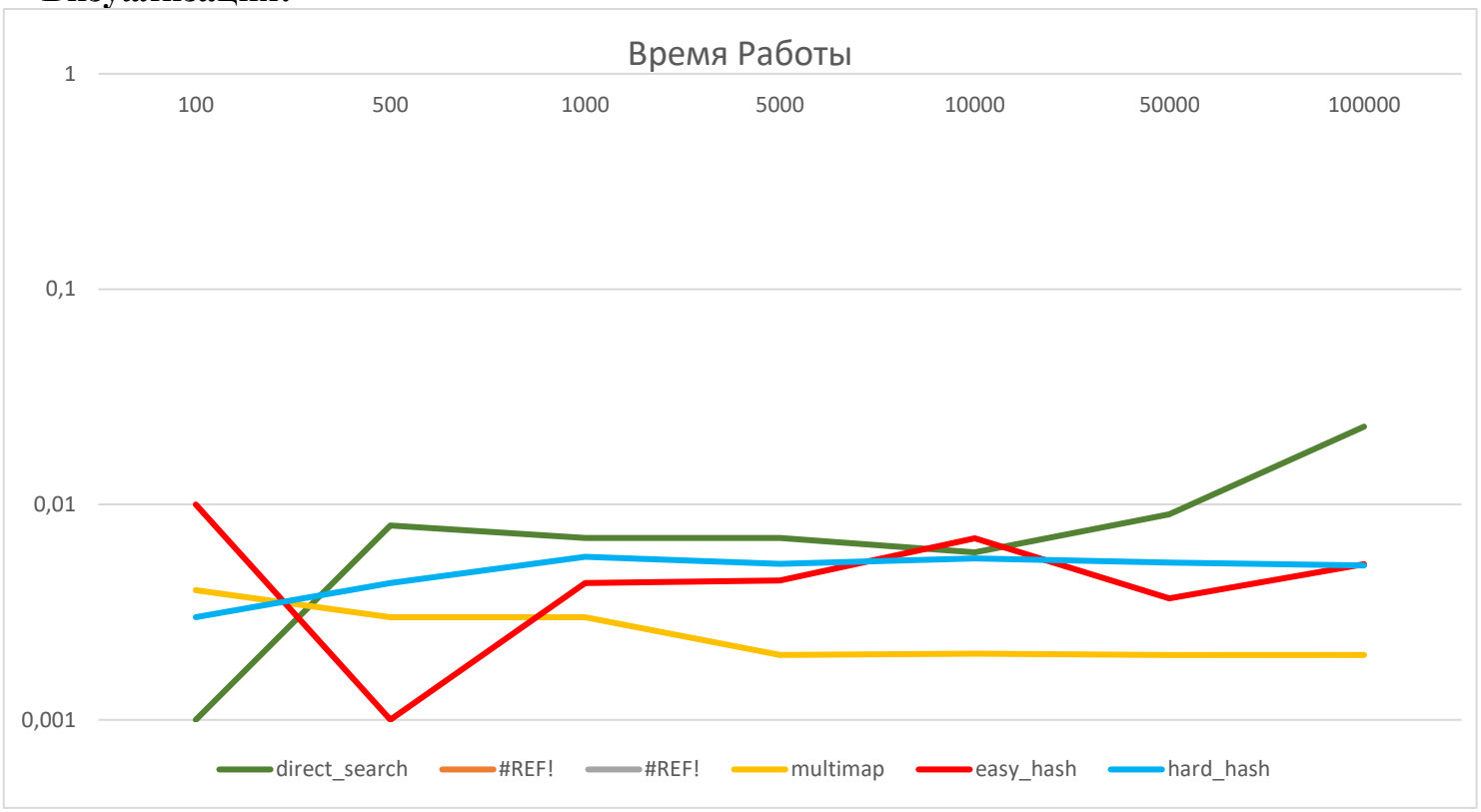
	100	500	1000	5000	10000	50000	100000
easy_hash	7	49	105	529	1042	5254	10354
hard_hash	4	36	61	356	710	3528	6943



Сравнение:

	100	500	1000	5000	10000	50000	100000
direct_search	0,001000404	0,007999638	0,006998041	0,00699885	0,005998756	0,008996964	0,022992849
multimap	0,003999617	0,002999855	0,002998663	0,002003742	0,002026392	0,002000404	0,002004457
easy_hash	0,010001898	0,001001358	0,004311323	0,0044384	0,006973267	0,003666878	0,005284786
hard_hash	0,002997866	0,004321089	0,005717745	0,005308857	0,005628338	0,00537466	0,005219927

Визуализация:



Вывод:

Результаты показывают, что что алгоритмы хеширования работают немного быстрее, чем прямой поиск.