САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №6
по курсу «Алгоритмы и структуры данных»
Тема: Хеширование. Хеш-таблицы
Вариант 17

Выполнил:
Останин А. С.
К3140

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №1. Множество

Реализуйте множество с операциями «добавление ключа», «удаление ключа», «проверка существования ключа».

```
from lab6.utils import read_data, write_data

class Set:
    """
    Класс множества
    """
    def __init__(self, _set):
        self._set = _set

    def hash_function(self, obj):
        """
        Хэш функция, которая считает сумму цифр и возвращает остаток от
деления на размер хеш-таблицы
        """
        return sum([int(i) for i in obj]) % len(self._set)

    def add(self, number):
        """
        Добавляет элемент в хеш-таблицу
        """
        name_id = self.hash_function(number)
        if [name_id, number] not in self._set[name_id]:
            self._set[name_id].append([name_id, number])

    def del_(self, number):
        """
        Удаляет элемент из хеш-таблицы
        """
        name_id = self.hash_function(number)
        for i in self._set[name_id]:
            if i[1] == number:
                self._set[name_id].remove(i)
                break

    def find_(self, number):
        """
        Ищет элемент в хеш-таблице
        """
        name_id = self.hash_function(number)
        for i in self._set[name_id]:
            if i[1] == number:
                return 'Y'
        return 'N'


def main(data):
    """
    Функция для решения задачи, которая работает с хеш-таблицей

    Пример входных данных: [['add', '123', 'John'], ['del', '22'], ['find',
'434']]
    Пример выходных данных: ['John', 'not found', 'not found']
    """
    book = [[] for _ in range(10)]
    result = []
```

```
    phone_book = Set(book)
    for i in data:
        if i[0] == 'A':
            phone_book.add(number=i[1])
        elif i[0] == 'D':
            phone_book.del_(number=i[1])
        elif i[0] == '?':
            result.append(phone_book.find_(number=i[1]))
    return result


def task1():
    PATH_INPUT = '../txtf/input.txt'
    PATH_OUTPUT = '../txtf/output.txt'

    data_split = [i.strip().split() for i in read_data(PATH_INPUT)[1:]]

    result = main(data_split)

    write_data(PATH_OUTPUT, '\n'.join(result))
    print(result)


if __name__ == "__main__":
    task1()
```

Напишем класс Set, который будет являться множеством, добавим в него методы hash-function, add, del_, find_, которые, соответственно, будут являться хеш-функцией, которая будет хешировать объекты, добавление элемента, удаление элемента и нахождения необходимого элемента.
В функции main будем считывать, обрабатывать и выводить данные.

```
Input:
8
A 2
A 5
A 3
? 2
? 4
A 2
D 2
? 2
Output:
Y
N
N
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	`9.809999028220773e-05 s`	`0.00061798095703125Mb`

Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: Реализована работа множества

Задача №2. Телефонная книга

В этой задаче ваша цель - реализовать простой менеджер телефонной книги. Он должен уметь обрабатывать следующие типы пользовательских запросов:

• add number name – это команда означает, что пользователь добавляет в телефонную книгу человека с именем name и номером телефона number. Если пользователь с таким номером уже существует, то ваш менеджер должен перезаписать соответствующее имя.

• del number – означает, что менеджер должен удалить человека с номером из телефонной книги. Если такого человека нет, то он должен просто игнорировать запрос.

• find number – означает, что пользователь ищет человека с номером телефона number. Менеджер должен ответить соответствующим именем или строкой «not found» (без кавычек), если такого человека в книге нет.

```
from lab6.utils import read_data, write_data

class PhoneBook:
    """
    Класс для телефонной книги, использующий хеш-таблицу для хранения
данных
    """
    def __init__(self, book):
        self.book = book

    def hash_function(self, obj):
        """
        Хэш функция, которая считает сумму цифр и возвращает остаток от
деления на размер хеш-таблицы
        """
        return sum([int(i) for i in obj]) % len(self.book)

    def add(self, number, name):
        """
        Добавляет запись в хеш-таблицу
        """
        name_id = self.hash_function(number)
        self.book[name_id] = [name, number]

    def del_(self, number):
        """
```

```
        Удаляет запись в хеш-таблице
        """
        name_id = self.hash_function(number)
        self.book[name_id] = ['', '']

    def find_(self, number):
        """
        Ищет запись в хеш-таблице
        """
        name_id = self.hash_function(number)
        if self.book[name_id] != ['', '']:
            return self.book[name_id][0]
        else:
            return 'not found'


def main(data):
    """
    Функция для решения задачи, которая работает с хеш-таблицей

    Пример входных данных: [['add', '123', 'John'], ['del', '22'], ['find',
'434']]
    Пример выходных данных: ['John', 'not found', 'not found']
    """
    book = [['', ''] for i in range(10)]
    result = []
    phone_book = PhoneBook(book)
    for i in data:
        if i[0] == 'add':
            phone_book.add(number=i[1], name=i[2])
        elif i[0] == 'del':
            phone_book.del_(number=i[1])
        elif i[0] == 'find':
            result.append(phone_book.find_(number=i[1]))
    return result


def task2():
    PATH_INPUT = '../txtf/input.txt'
    PATH_OUTPUT = '../txtf/output.txt'

    data_split = [i.strip().split() for i in read_data(PATH_INPUT)[1:]]

    result = main(data_split)

    write_data(PATH_OUTPUT, '\n'.join(result))
    print(result)


if __name__ == "__main__":
    task2()
```

Напишем класс PhoneBook, в котором напишем методы преобразования объекта с помощью хеш-функции, добавления контакта, удаления контакта и нахождения контакта

В функции main будем обрабатывать и возвращать данные. В функции task2() считывать и выводить данные.

```
Input:
12
add 911 police
add 76213 Mom
add 17239 Bob
find 76213
find 910
find 911
del 910
del 911
find 911
find 76213
add 76213 daddy
find 76213


Output:
Mom
not found
police
not found
Mom
daddy
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	0.0001306000049225986 s	0.001178741455078125 mb
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: решена задача на реализацию телефонной книги

Задача №5. Выборы в США

Как известно, в США президент выбирается не прямым голосованием, а путем двухуровневого голосования. Сначала проводятся выборы в каждом штате и определяется победитель выборов в данном штате. Затем проводятся государственные выборы: на этих выборах каждый штат имеет

определенное число голосов — число выборщиков от этого штата. На практике, все выборщики от штата голосуют в соответствии с результатами голосования внутри штата, то есть на заключительной стадии выборов в голосовании участвуют штаты, имеющие различное число голосов. Вам известно за кого проголосовал каждый штат и сколько голосов было отдано данным штатом. Подведите итоги выборов: для каждого из участника голосования определите число отданных за него голосов.

```
from lab6.utils import read_data, write_data


def hash_function(obj, size_hash_table):
    """
    Хеш-функция, которая считает сумму кодов символов и
    возвращает остаток от деления на размер хеш-таблицы

    Пример: hash_function('abc', 10) -> 4
    """
    sum_chars = sum([ord(i) for i in obj])
    return sum_chars % size_hash_table


def usa_elections(data, hash_table):
    """
    Заполняет хеш-таблицу и, используя хеш-функцию, вставляет в нее данные

    Пример входных данных: data=[['A', '1'], ['B', '2'], ['C', '3']]
                           hash_table=[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
    Пример выходных данных: hash_table=[[0, 'A', 1], [1, 'B', 2], [2, 'C',
3]]
    """
    size_hash_table = len(hash_table)
    for i in data:
        name_id = hash_function(i[0], size_hash_table)

        # если такой кандидат уже есть в хеш-таблице
        if hash_table[name_id] != [0, 0, 0]:

            # если такой кандидат уже есть в хеш-таблице
            if name_id == hash_table[name_id][0] and i[0] ==
hash_table[name_id][1]:
                hash_table[name_id][2] += int(i[1])
            else:

                # ищем свободное место, чтобы избавится от коллизии
                while hash_table[name_id] != [0, 0, 0]:
                    name_id = (name_id + 1) % size_hash_table
                else:
                    hash_table[name_id] = [name_id, i[0], int(i[1])]

        else:
            # добавляем новую запись в хеш-таблицу
            hash_table[name_id] = [name_id, i[0], int(i[1])]

    return hash_table


def main(data):
    """
    Функция для решения задачи, которая заполняет хеш-таблицу и возвращает
    список кандидатов и их количество голосов
```

```
    Пример входных данных: [['McCain', '10'], ['McCain', '5'], ['Obama',
'9'], ['Obama', '8'], ['McCain', '1']]
    Пример выходных данных: [['McCain', 16], ['Obama', 17]]
    """
    hash_table_ = [[0, 0, 0] for i in range(10)]
    usa_elections(data, hash_table_)
    candidates = []

    # находим уникальных кандидатов
    for i in data:
        name_id = hash_function(i[0], len(hash_table_))
        candidate = hash_table_[name_id][1]
        if candidate not in candidates:
            candidates.append(candidate)

    # находим количество голосов кандидатов
    for i in range(len(candidates)):
        name_id = hash_function(candidates[i], len(hash_table_))
        votes = hash_table_[name_id][2]
        candidates[i] = [candidates[i], votes]

    return sorted(candidates, key=lambda x: x[0])  # сортировка в
лексикографическом порядке


def task5():
    PATH_INPUT = '../txtf/input.txt'
    PATH_OUTPUT = '../txtf/output.txt'

    data_ = [i.strip().split() for i in read_data(PATH_INPUT)]
    result = main(data_)

    write_data(PATH_OUTPUT, '\n'.join([f'{i[0]} {i[1]}' for i in result]))
    print(result)


if __name__ == "__main__":
    task5()
```

Напишем функцию hash_function, которая будет хешировать объект. В функции usa_elections будем заполнять хеш-таблицу кандидатами и при повторении кандидата, будем суммировать число голосов за него. В функции main будем искать по хеш-таблице сколько голосов у каждого кандидата и записывать это в список candidates, после чего возвращать отсортированный в лексикографическом порядке фамилий кандидатов список candidates. В функции task5 будем считывать и обрабатывать входные данные, как и выводить полученные данные.

```
Input:
ivanov 100
ivanov 500
ivanov 300
petr 70
tourist 1
tourist 2
```

```
Output:
ivanov 900
petr 70
tourist 3
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	`0.00022519999765791 s`	`0.00140380859375 mb`
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: решена задача, используя хеш-функцию и хеш-таблицу

Вывод

Решены задачи, используя ООП и хеш-функции и хеш-таблицы.