САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №6

по курсу «Алгоритмы и структуры данных»

Тема: Хеширование. Хеш-таблицы

Вариант -

Выполнила:

Аксянова А.Р

К3140

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[Содержание отчета 2](#_Toc184773745)

[Задачи 3](#_Toc184773746)

[Задача №1. Множество 3](#_Toc184773747)

[Задача №2. Телефонная книга 4](#_Toc184773748)

[Задача №5. Выборы в сша 6](#_Toc184773749)

[Задача №7 . Драгоценные камни. 8](#_Toc184773750)

[Задача №8. Почти интерактивная хеш-таблица 10](#_Toc184773751)

[Результат работы всех задач 13](#_Toc184773752)

[Вывод 13](#_Toc184773753)

# Задачи

## Задача №1. Множество

1. Текст задачи

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

• **Формат входного файла (input.txt).** В первой строке входного файла находится строго положительное целое число операций *N*, не превышающее

5 · 105. В каждой из последующих *N* строк находится одна из следующих операций:

A *x* – добавить элемент *x* в множество. Если элемент уже есть в множестве, то ничего делать не надо.

D *x* – удалить элемент *x*. Если элемента *x* нет, то ничего делать не надо.

?*x*–еслиключ*x*естьвмножестве,выведите«Y»,еслинет,товыведите «N».

Аргументы указанных выше операций – **целые числа**, не превышающие по модулю 1018.

**Формат выходного файла (output.txt).** Выведите последовательно результат выполнения всех операций «?». Следуйте формату выходного файла из примера.

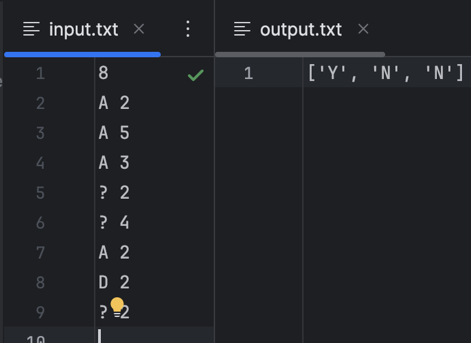
1. Листинг кода

import time  
import os  
from lab6.utils import inp, outp, caption  
  
CURRENT\_SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(CURRENT\_SCRIPT\_DIR)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
  
  
def process\_operations(n, operations):  
 result = []  
 set\_list = set()  
  
 for operation in operations:  
 op, x = operation[0], int(operation[1])  
  
 if op == 'A':  
 set\_list.add(x)  
 elif op == 'D':  
 set\_list.discard(x)  
 elif op == '?':  
 result.append('Y' if x in set\_list else 'N')  
  
 return result  
  
def task1():  
 n, s = inp(PATH\_INPUT, 0, 'task1')  
 res = str(process\_operations(n, s))  
 global task\_numb  
 task\_numb = 1  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

1. Текстовое объяснение решения.

Функция process\_operations принимает список операций, обрабатывает их и формирует результат. С помощью множества set она обеспечивает быстрый доступ к элементам. Функция поддерживает три команды: добавление числа в множество, удаление из него, и проверка существования числа.

1. Результат работы кода на примерах:



\*Результат работы кода (скрин терминала) так же будет приложен ниже

1. Вывод по задаче:

В результате выполнения задачи я научилась реализовывать множество

## Задача №2. Телефонная книга

1. Текст задачи

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 nameи 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. • Формат ввода / входного файла (input.txt). В первой строке входного файла содержится число *N* (1 ≤ *N* ≤ 105) - количество запросов. Далее следуют *N* строк, каждая из которых содержит один запрос в формате, описанном выше.

Все номера телефонов состоят из десятичных цифр, в них нет нулей в начале номера, и каждый состоит не более чем из 7 цифр. Все имена представляют собой непустые строки из латинских букв, каждая из которых имеет длину не более 15. Гарантируется при проверке, что не будет человека с именем «not found».

Формат вывода / выходного файла (output.txt). Выведите результат каждого поискового запроса find – имя, соответствующее номеру телефона, или «not found» (без кавычек), если в телефонной книге нет человека с таким номером телефона. Выведите по одному результату в каждой строке в том же порядке, как были заданы запросы типа find во входных данных.

1. Листинг кода

import time  
import os  
from lab6.utils import inp, outp, caption  
  
CURRENT\_SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(CURRENT\_SCRIPT\_DIR)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
  
  
def Phonebook(n, operations):  
 phonebook = {}  
 result = []  
  
 for operation in operations:  
 if operation[0] == "add":  
 number, name = operation[1], operation[2]  
 phonebook[number] = name  
 elif operation[0] == "del":  
 number = operation[1]  
 phonebook.pop(number, None)  
 elif operation[0] == "find":  
 number = operation[1]  
 result.append(phonebook.get(number, "not found"))  
  
 return result  
  
  
def task1():  
 n, s = inp(PATH\_INPUT, 0, 'task2')  
 res = str(Phonebook(n, s))  
 global task\_numb  
 task\_numb = 2  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

1. Текстовое объяснение решения.

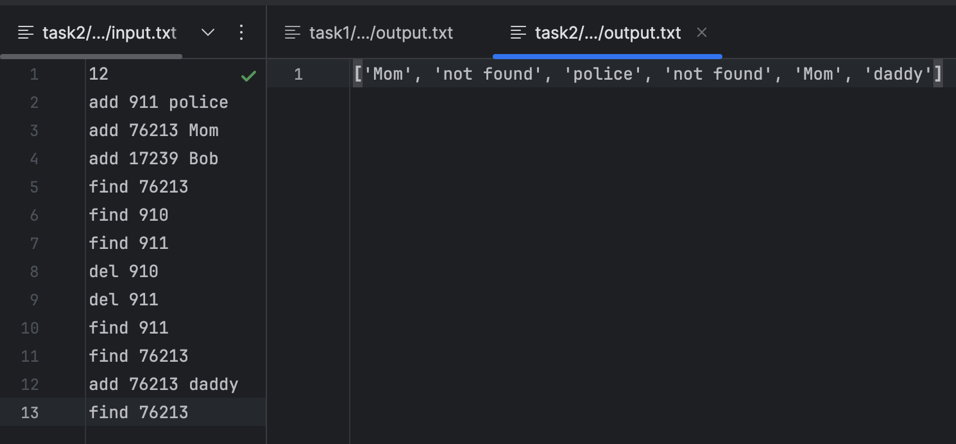
Функция Phonebook количество операций и список команд для обработки. Телефонная книга хранится в виде словаря (dict), где ключами являются номера телефонов, а значениями — имена их владельцев. Каждая команда обрабатывается в цикле:

**Добавление записи (**add**)**: Если команда предполагает добавление, номер телефона и имя сохраняются в словаре. Если номер уже существует, его значение обновляется новым именем.

**Удаление записи (**del**)**: Для удаления записи используется метод pop, который удаляет номер телефона из словаря. Если номера нет, операция просто игнорируется.

**Поиск (**find**)**: Если пользователь запрашивает поиск номера телефона, программа проверяет его наличие в словаре. Если номер найден, возвращается соответствующее имя, иначе возвращается строка "not found".

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я научилась реализовать простой менеджер телефонной книги.

## Задача №5. Выборы в сша

1. Текст задачи

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 результами 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.

* **Форматввода/входногофайла(input.txt).**Каждаястрокавходногофайла содержит фамилию кандидата, за которого отдают голоса выборщики этого штата, затем через пробел идет количество выборщиков, отдавших голоса за этого кандидата.
* **Формат вывода / выходного файла (output.txt).** Выведите фамилии всех кандидатов в ***лексикографическом*** порядке, затем, через пробел, количество отданных за них голосов.

1. Листинг кода

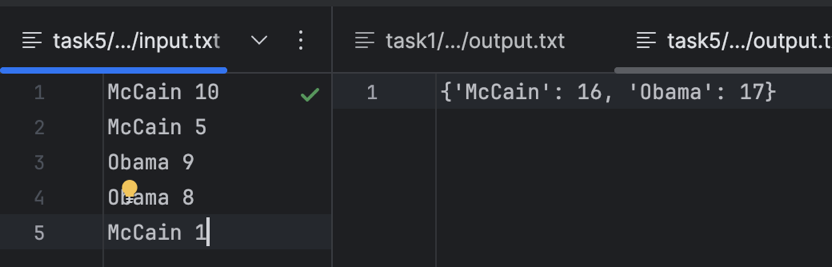
import time  
import os  
from lab6.utils import inp, outp, caption  
  
CURRENT\_SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(CURRENT\_SCRIPT\_DIR)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
  
  
def count\_votes(s):  
 votes = {}  
  
 for i in range(len(s)):  
 parts = s[i].split()  
 candidate\_name = parts[0]  
 number\_of\_votes = int(parts[1])  
  
 if candidate\_name in votes:  
 votes[candidate\_name] += number\_of\_votes  
 else:  
 votes[candidate\_name] = number\_of\_votes  
  
 return votes  
  
  
def task1():  
 s = inp(PATH\_INPUT, 0, 'task5')  
 res = str(count\_votes(s))  
 global task\_numb  
 task\_numb = 5  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

elapsed\_time = time.perf\_counter() - start  
 print(caption(task\_numb, elapsed\_time))

1. Текстовое объяснение решения.

Функция count\_votes создает словарь votes, где ключами являются имена кандидатов, а значениями — общее количество голосов, собранных каждым кандидатом. Для обработки данных функция проходит по каждой строке списка, разделяет строку на имя кандидата и количество голосов, а затем обновляет соответствующую запись в словаре. Если кандидат уже присутствует в словаре, к его текущему количеству голосов добавляется новое значение. Если же кандидат появляется впервые, он добавляется в словарь с начальным количеством голосов.

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я написала алгоритм подсчета голосов на выборах.

## Задача №7 . Драгоценные камни.

1. Текст задачи

В одной далекой восточной стране до сих пор по пустыням ходят караваны верблюдов, с помощью которых купцы перевозят пряности, драгоценности и дорогие ткани. Разумеется, основная цель купцов состоит в том, чтобы подороже продать имеющийся у них товар. Недавно один из караванов прибыл во дворец одного могущественного шаха.

Купцы хотят продать шаху n драгоценных камней, которые они привезли с собой. Для этого они выкладывают их перед шахом в ряд, после чего шах оценивает эти камни и принимает решение о том, купит он их или нет. Видов драгоценных камней на Востоке известно не очень много всего 26, поэтому мы будем обозначать виды камней с помощью строчных букв латинского алфавита. Шах обычно оценивает камни следующим образом. Он заранее определил несколько упорядоченных пар типов камней: (*a*1*,b*1), (*a*2*,b*2), ..., (*ak,bk*). Эти пары он называет красивыми, их множество мы обозначим как *P*. Теперь представим ряд камней, которые продают купцы, в виде строки *S* длины *n* из строчных букв латинского алфавита. Шах считает число таких пар (*i,j*), что 1 ≤ *i < j* ≤ *n*, а камни *Si* и *Sj* образуют красивую пару, то есть существует такое число 1 ≤ *q* ≤ *k*, что *Si* = *aq* и *Sj* = *bq*.

Если число таких пар оказывается достаточно большим, то шах покупает все камни. Однако в этот раз купцы привезли настолько много камней, что шах не может посчитать это число. Поэтому он вызвал своего визиря и поручил ему этот подсчет. Напишите программу, которая находит ответ на эту задачу. • **Форматввода/входногофайла(input.txt).**Перваястрокавходногофайла содержит целые числа *n* и *k* (1 ≤ *n* ≤ 100000, 1 ≤ *k* ≤ 676) – число камней, которые привезли купцы и число пар, которые шах считает красивыми. Вторая строка входного файла содержит строку *S*, описывающую типы камней, которые привезли купцы.

Далее следуют *k* строк, каждая из которых содержит две строчных буквы латинского алфавита и описывает одну из красивых пар камней.

* **Формат вывода / выходного файла (output.txt).** В выходной файл выведите ответ на задачу – количество пар, которое должен найти визирь.

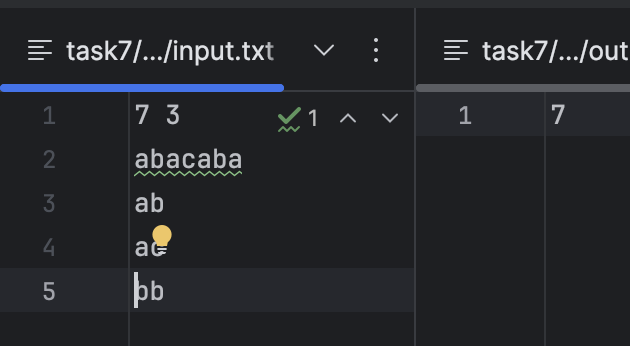
1. Листинг кода

import time  
import os  
from lab6.utils import inp, outp, caption  
  
CURRENT\_SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(CURRENT\_SCRIPT\_DIR)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
  
  
def Precious\_stones(n, k, s, pairs):  
 beautiful\_pairs = {}  
 for a, b in pairs:  
 if a not in beautiful\_pairs:  
 beautiful\_pairs[a] = set()  
 beautiful\_pairs[a].add(b)  
  
 stone\_count = {}  
 result = 0  
  
 for i in range(n):  
 current\_stone = s[i]  
  
 if current\_stone in beautiful\_pairs:  
 for b in beautiful\_pairs[current\_stone]:  
 if b in stone\_count:  
 result += stone\_count[b]  
  
 if current\_stone in stone\_count:  
 stone\_count[current\_stone] += 1  
 else:  
 stone\_count[current\_stone] = 1  
  
 return result  
  
def task1():  
 n, k, s, l = inp(PATH\_INPUT, 0, 'task7')  
 res = str(Precious\_stones(n, k, s, l))  
 global task\_numb  
 task\_numb = 7  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

1. Текстовое объяснение решения.

Функция Precious\_stones преобразует список пар в удобную структуру данных: словарь beautiful\_pairs, где каждая буква-камень ключа ассоциируется с множеством букв, образующих с ней "красивую" пару. Затем программа использует этот словарь для анализа строки камней. Для подсчета "красивых" пар программа проходит по строке камней и для каждого камня проверяет, сколько подходящих пар можно составить с уже встреченными камнями, используя словарь stone\_count, который отслеживает частоту появления каждого типа камня. Если текущий камень имеет "красивые" пары, программа увеличивает результат на количество подходящих камней, которые уже встречались.

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я написала алгоритм для решения задачи.

## Задача №8. Почти интерактивная хеш-таблица

1. Текст задачи

В данной задаче у Вас не будет проблем ни с вводом, ни с выводом. Просто реализуйте быструю хеш-таблицу.

В этой хеш-таблице будут храниться целые числа из диапазона [0;1015 − 1]. Требуется поддерживать добавление числа *x* и проверку того, есть ли в таблице число *x*. Числа, с которыми будет работать таблица, генерируются следующим образом. Пусть имеется четыре целых числа *N,X,A,B* такие что:

* 1 ≤ *N* ≤ 107
* 1 ≤ *X* ≤ 1015
* 1 ≤ *A* ≤ 103
* 1 ≤ *B* ≤ 1015

Требуется *N* раз выполнить следующую последовательность операций:

* Если *X* содержится в таблице, то установить *A* ← (*A* + *AC*) mod 103, *B* ← (*B* + *BC*) mod 1015.
* Если *X* не содержится в таблице, то добавить *X* в таблицу и установить

*A* ← (*A* + *AD*) mod 103, *B* ← (*B* + *BD*) mod 1015.

* Установить *X* ← (*X* · *A* + *B*) mod 1015.

Начальные значения *X,A* и *B*, а также *N,AC,BC,AD* и *BD* даны во входном файле. Выведите значения *X,A* и *B* после окончания работы.

* **Формат входного файла (input.txt).** В первой строке входного файла содержится четыре целых числа *N,X,A,B*. Во второй строке содержится еще четыре целых числа *AC,BC,AD* и *BD* такие что 0 ≤ *AC,AD <* 103, 0 ≤ *BC,BD <* 1015.
* **Формат выходного файла (output.txt).** Выведите значения *X,A* и *B* после окончания работы.

1. Листинг кода

import time  
import os  
from lab6.utils import inp, outp, caption  
  
CURRENT\_SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(CURRENT\_SCRIPT\_DIR)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
  
  
def hash\_table(N, X, A, B, AC, BC, AD, BD):  
 table = set()  
  
 for \_ in range(N):  
 if X in table:  
 A = (A + AC) % 103  
 B = (B + BC) % 10 \*\* 15  
 else:  
 table.add(X)  
 A = (A + AD) % 103  
 B = (B + BD) % 10 \*\* 15  
  
 X = (X \* A + B) % 10 \*\* 15  
  
 return X, A, B  
  
def task1():  
 n, x, a, b, A\_c, B\_c, A\_d, B\_d = inp(PATH\_INPUT, 0, 'task8')  
 res = str(hash\_table(n, x, a, b, A\_c, B\_c, A\_d, B\_d))  
 global task\_numb  
 task\_numb = 8  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption(task\_numb, time))

1. Текстовое объяснение решения.

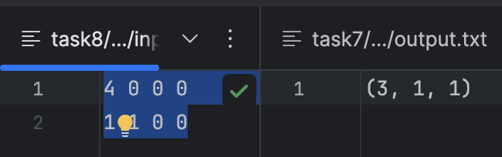
Таблица представлена в виде множества table, которое используется для проверки наличия чисел. На каждой итерации выполняется одна из двух ветвей:

Если текущее значение X уже присутствует в таблице, параметры A и B изменяются по формулам

Если X отсутствует в таблице, оно добавляется в таблицу, а A и изменяются по другим формулам

После этого значение X пересчитывается с использованием обновленных A и B по третьей формуле. Этот процесс повторяется N раз, и в итоге возвращаются финальные значения

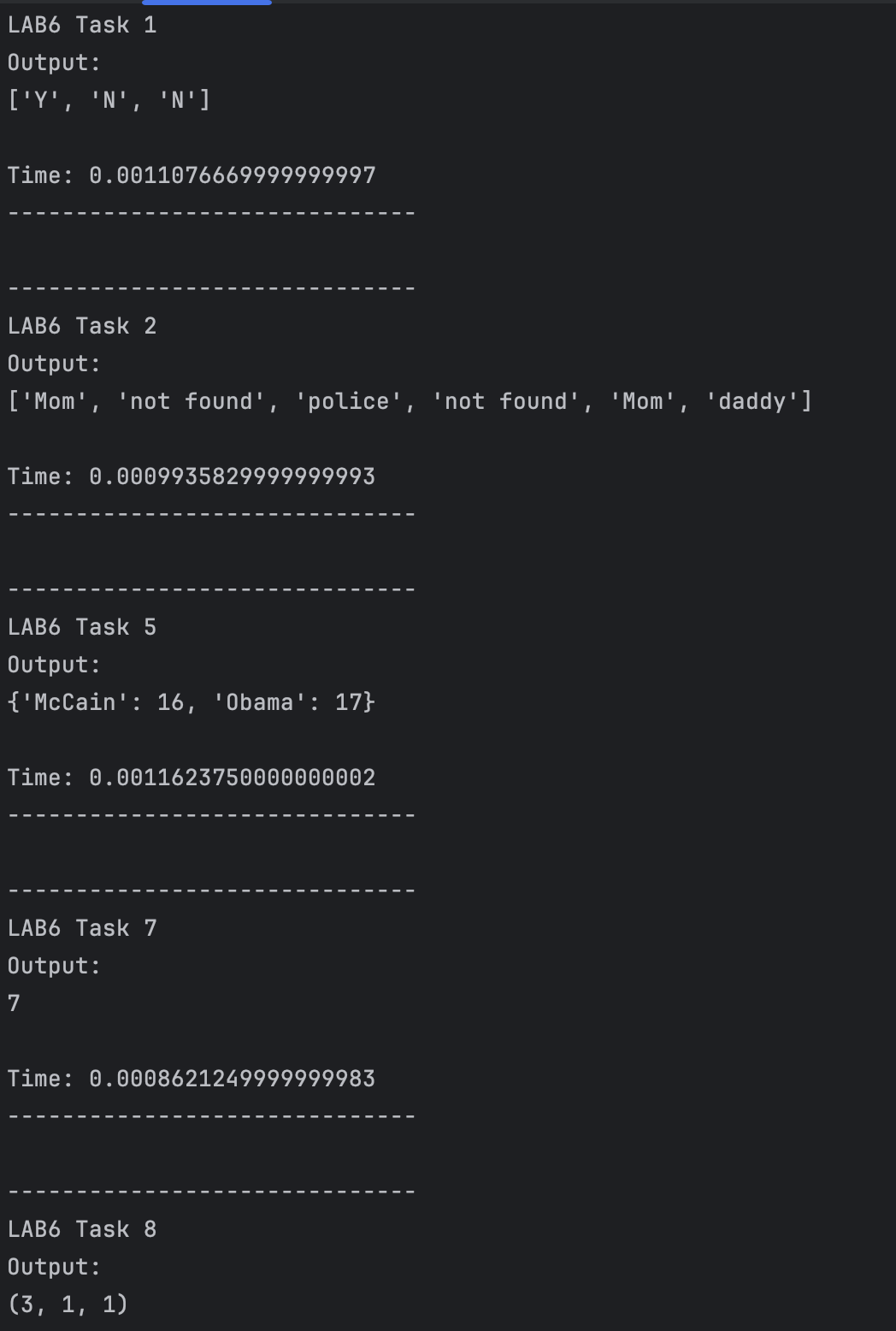
1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я реализовала быструю хеш-таблицу

## Результат работы всех задач

****

# Вывод

В результате лабораторной я изучита хеширование