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 9

Выполнила:

Коновалова Кира Романовна

К3139

Проверил:

Афанасьев А.В

Санкт-Петербург

2024 г.

# Содержание отчета

**[Задачи по варианту. Задачи по выбору 3](#_Toc17848)**

[Задача №1. Множество (по выбору) 3](#_Toc23193)

[Задача №2. Телефонная книга (по выбору) 6](#_Toc985)

[Задача №4. Прошитый ассоциативный массив 9](#_Toc23328)

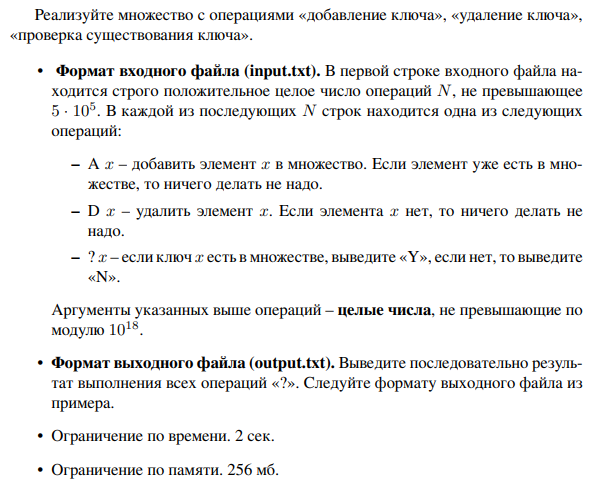
[Задача №5. Выборы в США 14](#_Toc31783)

**[Вывод 18](#_Toc1203)**

# Задачи по варианту. Задачи по выбору

## Задача №1. Множество (по выбору)

Текст задачи:



Листинг кода:

import os  
import sys  
  
def process\_operations(operations):  
 *"""  
 Выполняет операции над множеством.  
 :param operations: Список строк операций  
 :return: Список результатов для операций "? x"  
 """* result = []  
 custom\_set = set()  
  
 for operation in operations:  
 op\_type, x = operation.split()  
 x = int(x)  
  
 if op\_type == "A":  
 custom\_set.add(x)  
 elif op\_type == "D":  
 custom\_set.discard(x)  
 elif op\_type == "?":  
 result.append("Y" if x in custom\_set else "N")  
  
 return result  
  
  
def main(input\_data, output\_file):  
 *"""  
 Основная функция для запуска задачи.  
 :param input\_data: Строка с содержимым входного файла  
 :param output\_file: Объект файла для записи вывода  
 """* lines = input\_data.strip().split("\n")  
 n = int(lines[0])  
 operations = lines[1:]  
  
 # Обрабатываем операции  
 results = process\_operations(operations)  
  
 # Записываем результаты в файл  
 output\_file.write("\n".join(results) + "\n")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

Текстовое объяснение задачи:

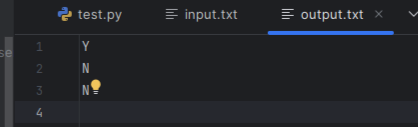
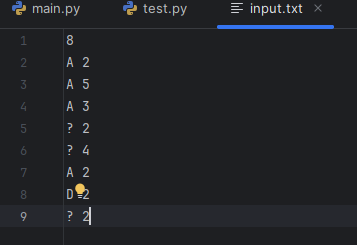
Для выполнения задачи использовалась структура данных set, встроенная в Python, которая обеспечивает эффективное выполнение операций добавления, удаления и проверки существования элемента.

Входные данные считывались из файла, а результаты для запросов типа «? x» записывались в выходной файл. Каждый запрос обрабатывался в цикле, и операции выполнялись с использованием методов add(), discard(), а для проверки существования элемента использовался оператор in

Тесты:

import unittest  
from lab6.task1.src.main import process\_operations  
  
class TestSetOperations(unittest.TestCase):  
  
 def test\_add\_and\_query(self):  
 operations = ["A 1", "A 2", "? 1", "? 3"]  
 expected = ["Y", "N"]  
 self.assertEqual(process\_operations(operations), expected)  
  
 def test\_add\_duplicate\_and\_query(self):  
 operations = ["A 1", "A 1", "? 1", "? 2"]  
 expected = ["Y", "N"]  
 self.assertEqual(process\_operations(operations), expected)  
  
 def test\_remove\_and\_query(self):  
 operations = ["A 1", "A 2", "D 1", "? 1", "? 2"]  
 expected = ["N", "Y"]  
 self.assertEqual(process\_operations(operations), expected)  
  
 def test\_remove\_nonexistent(self):  
 operations = ["A 1", "D 2", "? 1", "? 2"]  
 expected = ["Y", "N"]  
 self.assertEqual(process\_operations(operations), expected)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

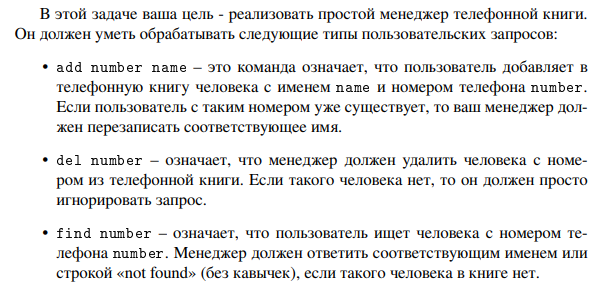


Вывод по задаче:

В этой задаче я реализовала множество с операциями добавления, удаления и проверки существования элементов. Использование встроенной структуры данных set в Python позволило эффективно выполнять эти операции с временной сложностью O(1) в среднем, что обеспечило быструю обработку большого количества запросов

## Задача №2. Телефонная книга (по выбору)

Текст задачи:



Листинг кода:

import os  
import sys  
  
def process\_queries(input\_data):  
 phonebook = {}  
 result = []  
 for line in input\_data.strip().split("\n"):  
 parts = line.split()  
 command = parts[0]  
  
 if command == "add":  
 number = parts[1]  
 name = parts[2]  
 phonebook[number] = name  
  
 elif command == "del":  
 number = parts[1]  
 if number in phonebook:  
 del phonebook[number]  
  
 elif command == "find":  
 number = parts[1]  
 if number in phonebook:  
 result.append(phonebook[number])  
 else:  
 result.append("not found")  
 return result  
  
def main(input\_data, output\_file):  
 results = process\_queries(input\_data)  
 # Записываем результаты в выходной файл  
 output\_file.write("\n".join(results) + "\n")  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

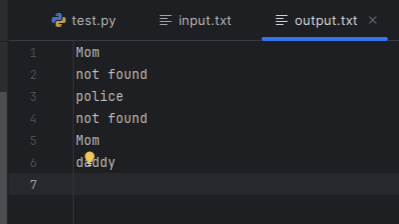
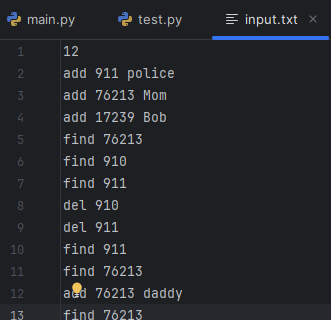
Текстовое объяснение задачи:

Для реализации использовался словарь, где ключом является номер телефона, а значением — имя владельца. Каждая операция выполнялась по аналогии с использованием стандартных методов словаря (add, del, get)

Тесты:

import unittest  
from io import StringIO  
from lab6.task2.src.main import main  
class TestPhonebook(unittest.TestCase):  
 def setUp(self):  
 # Метод setUp() будет запускаться перед каждым тестом  
 self.input\_data = ""  
 self.output = StringIO()  
  
 def run\_main(self, input\_data):  
 *"""Запуск основного алгоритма и возврат результата"""* self.input\_data = input\_data  
 # Используем StringIO в качестве файла, не закрываем его сразу  
 output\_file = StringIO()  
 main(self.input\_data, output\_file)  
 return output\_file.getvalue().strip()  
  
 def test\_add\_find(self):  
 input\_data = """5  
add 1234567 John  
add 2345678 Alice  
find 1234567  
find 2345678  
find 9876543"""  
 expected\_output = """John  
Alice  
not found"""  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
 def test\_del(self):  
 input\_data = """4  
add 1234567 John  
add 2345678 Alice  
del 1234567  
find 1234567  
find 2345678"""  
 expected\_output = """not found  
Alice"""  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_multiple\_add(self):  
 input\_data = """3  
add 1234567 John  
add 1234567 Alice  
find 1234567"""  
 expected\_output = """Alice"""  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
 def test\_edge\_case\_empty(self):  
 input\_data = """1  
find 1234567"""  
 expected\_output = """not found"""  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

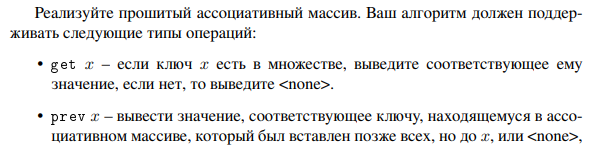


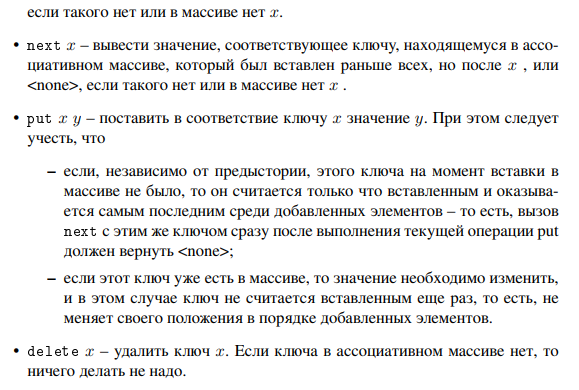
Вывод по задаче:

В данной задаче был реализован простой менеджер телефонной книги с операциями добавления, удаления и поиска. Использование словаря для хранения записей позволило обеспечить быструю обработку операций с временной сложностью O(1), что сделало решение эффективным для больших объемов данных

## Задача №4. Прошитый ассоциативный массив

Текст задачи:





Листинг кода:

import os  
import sys  
  
class ThreadedMap:  
 def \_\_init\_\_(self):  
 self.map = {}  
 self.order = []  
  
 def put(self, key, value):  
 if key not in self.map:  
 self.order.append(key)  
 self.map[key] = value  
  
 def delete(self, key):  
 if key in self.map:  
 del self.map[key]  
 self.order.remove(key)  
  
 def get(self, key):  
 return self.map.get(key, "<none>")  
  
 def prev(self, key):  
 if key not in self.map:  
 return "<none>"  
 idx = self.order.index(key)  
 return self.map[self.order[idx - 1]] if idx > 0 else "<none>"  
  
 def next(self, key):  
 if key not in self.map:  
 return "<none>"  
 idx = self.order.index(key)  
 return self.map[self.order[idx + 1]] if idx < len(self.order) - 1 else "<none>"  
  
  
def process\_operations(operations):  
 *"""  
 Выполняет операции над прошитым ассоциативным массивом.  
 :param operations: Список строк операций  
 :return: Список результатов для операций "get", "prev", "next"  
 """* result = []  
 threaded\_map = ThreadedMap()  
  
 for operation in operations:  
 parts = operation.split()  
 command, key = parts[0], parts[1]  
  
 if command == "put":  
 value = parts[2]  
 threaded\_map.put(key, value)  
 elif command == "delete":  
 threaded\_map.delete(key)  
 elif command == "get":  
 result.append(threaded\_map.get(key))  
 elif command == "prev":  
 result.append(threaded\_map.prev(key))  
 elif command == "next":  
 result.append(threaded\_map.next(key))  
  
 return result  
  
  
def main(input\_data, output\_file):  
 *"""  
 Основная функция для запуска задачи.  
 :param input\_data: Строка с содержимым входного файла  
 :param output\_file: Объект файла для записи вывода  
 """* lines = input\_data.strip().split("\n")  
 n = int(lines[0])  
 operations = lines[1:]  
  
 # Обрабатываем операции  
 results = process\_operations(operations)  
  
 # Записываем результаты в файл  
 output\_file.write("\n".join(results) + "\n")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

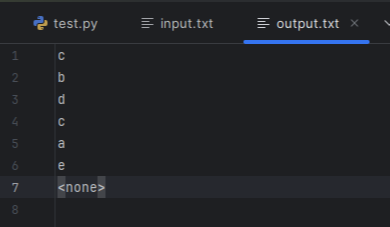
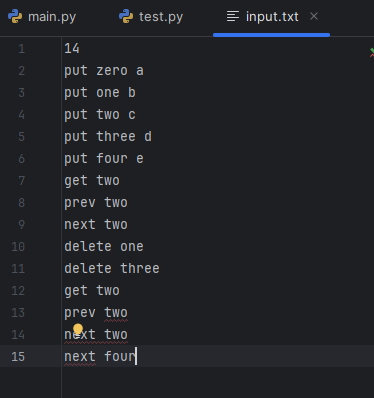
Текстовое объяснение задачи:

Для реализации использовалась структура данных dict для хранения значений, а также список order для отслеживания порядка добавления ключей. Это позволяет эффективно выполнять операции prev и next, учитывая порядок вставки

Тесты:

import unittest  
from lab6.task4.src.main import ThreadedMap  
  
class TestThreadedMap(unittest.TestCase):  
  
 def test\_put\_and\_get(self):  
 tm = ThreadedMap()  
 tm.put("a", "1")  
 tm.put("b", "2")  
 self.assertEqual(tm.get("a"), "1")  
 self.assertEqual(tm.get("b"), "2")  
 self.assertEqual(tm.get("c"), "<none>")  
  
 def test\_update\_put(self):  
 tm = ThreadedMap()  
 tm.put("a", "1")  
 tm.put("a", "2")  
 self.assertEqual(tm.get("a"), "2")  
  
 def test\_delete(self):  
 tm = ThreadedMap()  
 tm.put("a", "1")  
 tm.put("b", "2")  
 tm.delete("a")  
 self.assertEqual(tm.get("a"), "<none>")  
 self.assertEqual(tm.get("b"), "2")  
  
 def test\_prev(self):  
 tm = ThreadedMap()  
 tm.put("a", "1")  
 tm.put("b", "2")  
 tm.put("c", "3")  
 self.assertEqual(tm.prev("b"), "1")  
 self.assertEqual(tm.prev("a"), "<none>")  
  
 def test\_next(self):  
 tm = ThreadedMap()  
 tm.put("a", "1")  
 tm.put("b", "2")  
 tm.put("c", "3")  
 self.assertEqual(tm.next("b"), "3")  
 self.assertEqual(tm.next("c"), "<none>")  
  
 def test\_delete\_and\_order(self):  
 tm = ThreadedMap()  
 tm.put("a", "1")  
 tm.put("b", "2")  
 tm.put("c", "3")  
 tm.delete("b")  
 self.assertEqual(tm.prev("c"), "1")  
 self.assertEqual(tm.next("a"), "3")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

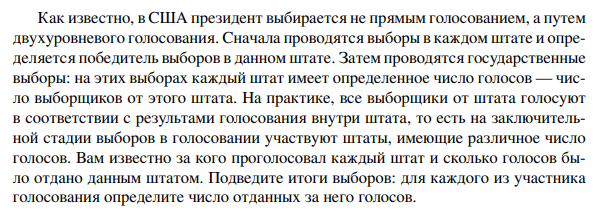


Вывод по задаче:

В этой задаче была реализована структура данных, поддерживающая операции с учетом порядка вставки ключей. С помощью словаря для хранения значений и списка для отслеживания порядка добавления ключей, удалось эффективно реализовать все требуемые операции с учетом их порядка

## Задача №5. Выборы в США

Текст задачи:



Листинг кода:

import os  
import sys  
from collections import defaultdict  
  
def calculate\_votes(input\_lines):  
 *"""  
 Подсчитывает количество голосов для каждого кандидата.  
 :param input\_lines: Список строк входного файла  
 :return: Словарь с подсчетом голосов {кандидат: количество голосов}  
 """* vote\_count = defaultdict(int)  
 for line in input\_lines:  
 candidate, votes = line.rsplit(maxsplit=1)  
 vote\_count[candidate] += int(votes)  
 return vote\_count  
  
def main(input\_data, output\_file):  
 *"""  
 Основная функция для запуска задачи.  
 :param input\_data: Строка с содержимым входного файла  
 :param output\_file: Объект файла для записи вывода  
 """* lines = input\_data.strip().split("\n")  
  
 # Подсчитываем голоса  
 vote\_count = calculate\_votes(lines)  
  
 # Сортируем кандидатов в лексикографическом порядке и записываем результат  
 for candidate in sorted(vote\_count):  
 output\_file.write(f"{candidate} {vote\_count[candidate]}\n")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

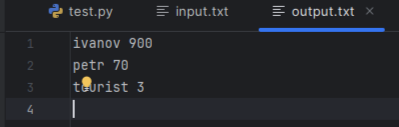
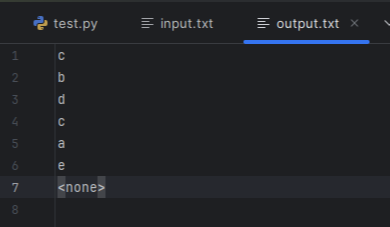
Текстовое объяснение задачи:

Необходимо подсчитать количество голосов, отданных каждому кандидату в выборах США, где каждый штат имеет определенное количество голосов.ля подсчета голосов использовался словарь, где ключом является имя кандидата, а значением — количество голосов. Для корректного подсчета голосов из каждого штата использовался метод defaultdict из библиотеки collections

Тесты:

import unittest  
from lab6.task5.src.main import calculate\_votes  
  
class TestCalculateVotes(unittest.TestCase):  
  
 def test\_single\_candidate(self):  
 input\_lines = [  
 "Smith 10",  
 "Smith 20",  
 "Smith 30"  
 ]  
 expected = {"Smith": 60}  
 self.assertEqual(calculate\_votes(input\_lines), expected)  
  
 def test\_multiple\_candidates(self):  
 input\_lines = [  
 "Smith 10",  
 "Johnson 15",  
 "Smith 20",  
 "Johnson 25",  
 "Doe 5"  
 ]  
 expected = {  
 "Smith": 30,  
 "Johnson": 40,  
 "Doe": 5  
 }  
 self.assertEqual(calculate\_votes(input\_lines), expected)  
  
 def test\_no\_votes(self):  
 input\_lines = []  
 expected = {}  
 self.assertEqual(calculate\_votes(input\_lines), expected)  
  
 def test\_lexicographical\_order(self):  
 input\_lines = [  
 "Zane 50",  
 "Alice 30",  
 "Bob 20"  
 ]  
 result = calculate\_votes(input\_lines)  
 sorted\_candidates = sorted(result.keys())  
 self.assertEqual(sorted\_candidates, ["Alice", "Bob", "Zane"])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

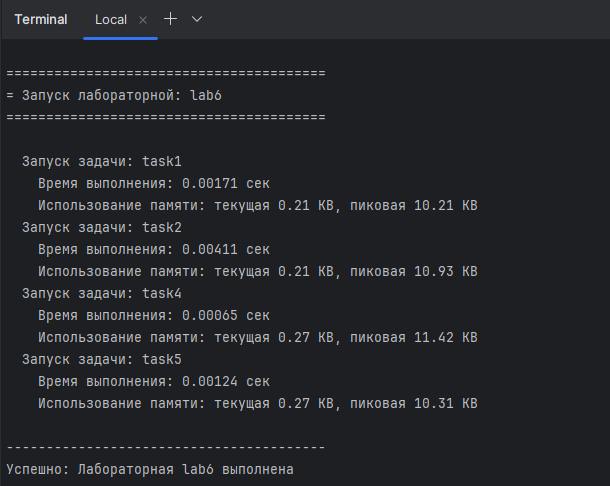
Результат работы кода на примерах из текста задачи:



Вывод по задаче:

В задаче по подсчету голосов для кандидатов использовался словарь для хранения количества голосов, отданных за каждого кандидата. Этот подход обеспечил точный и быстрый подсчет голосов, при этом сортировка кандидатов в лексикографическом порядке позволила корректно вывести результаты выборов.

**Запуск всех задач:**

****

# Вывод

В ходе выполнения лабораторной работы были реализованы основные операции для работы с хеш-таблицами и ассоциативными массивами. Для каждой задачи использовались подходящие структуры данных, такие как множества и словари, которые обеспечивают эффективное выполнение операций. Важной частью работы было использование стандартных функций Python для обработки входных данных и записи результатов