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 20

Выполнила:
Толстухина К.А.
К3139

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 год

Оглавление

Задачи по варианту

Задача №1. Множество

Текст задачи

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

Листинг кода

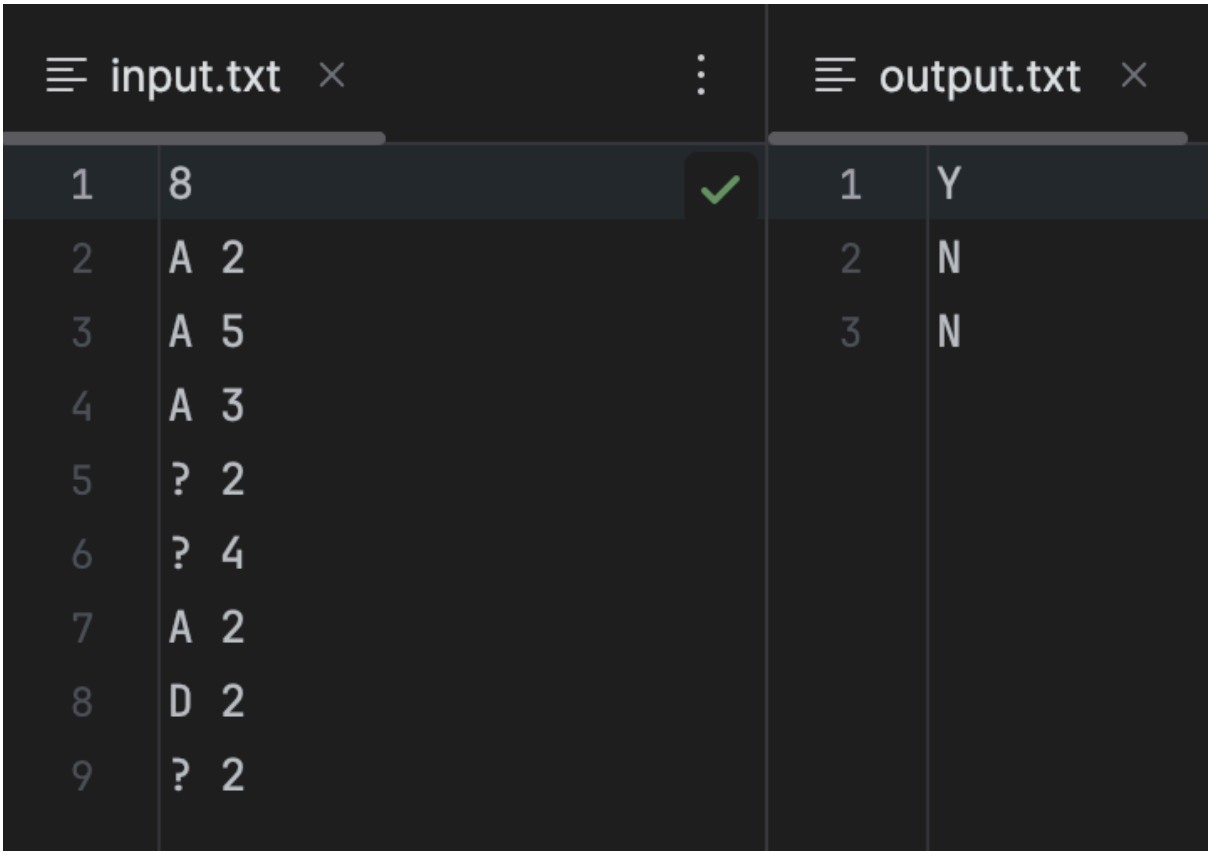
```
import tracemalloc
import time
from lab6.utils import *

t_start = time.perf_counter()
tracemalloc.start()


def Set(operations: list[str]) -> list[str]:
    """
    реализация работы множества
    :param operations: list[str]
    :return: list[str]
    """
    data = set()
    res = []
    for operation in operations:
        if operation.startswith("A"):
            data.add(operation.split()[1])
        elif operation.startswith("D"):
            data.remove(operation.split()[1])
        else:
            if operation.split()[1] in data:
                res.append("Y")
            else:
                res.append("N")
    return res
```

```
CURRENT_DIR                                                    =
os.path.dirname(os.path.abspath(__file__))
TXTF_DIR                                                       =
os.path.join(os.path.dirname(CURRENT_DIR), "txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")

if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
        commands  =  [command.strip()  for  command  in
lines[1:]]
    if 1 <= len(commands) <= 5 * 10 ** 5:
        result = Set(commands)
        write_file('\n'.join(result), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

            print("Время   работы:   %s   секунд"   %
(time.perf_counter() - t_start))
                    print("Затрачено       памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()
```

Текстовое объяснение

Подключаю две библиотеки для отслеживания памяти и времени. Затем Функция для работы с множеством, в которую я передаю список команд. Далее в цикле я считываю команду и в зависимости от нее добавляю/удаляю/проверяю наличие элемента в множестве. Далее вне функции я прописываю пути к необходимым директориям/файлам. И в основной части, я считываю данные, проверяю их валидность, вызываю функцию и вывожу время и память.

Пример работы(скрин файлов)



	Время	Память
пример	0.0008872080070432276 секунд	15107 байт

Вывод: была реализована работа с множеством

Задача №2. Телефонная книга

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 пате 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.

Листинг кода

```
class PhoneBook:
    """
    Класс для реализации работы телефонной книги
    """
    def __init__(self) -> None:
        self.phone_book = dict()

    def add_person(self, name: str, number: int) ->
None:
        """
        функция для добавления человека
        :param name: str
        :param number: int
        :return: None
        """
        self.phone_book[number] = name

    def del_number(self, number: int) -> None:
        """
        функция для удаления человека по номеру
        :param number: int
        :return: None
        """
        if number in self.phone_book.keys():
            del self.phone_book[number]
```

```
    def find_number(self, number: int) -> str:
        """
        функция для нахождения человека по номеру
        :param number: int
        :return: str
        """
        if number in self.phone_book.keys():
            return self.phone_book[number]
        return "not found"
```

Текстовое объяснение

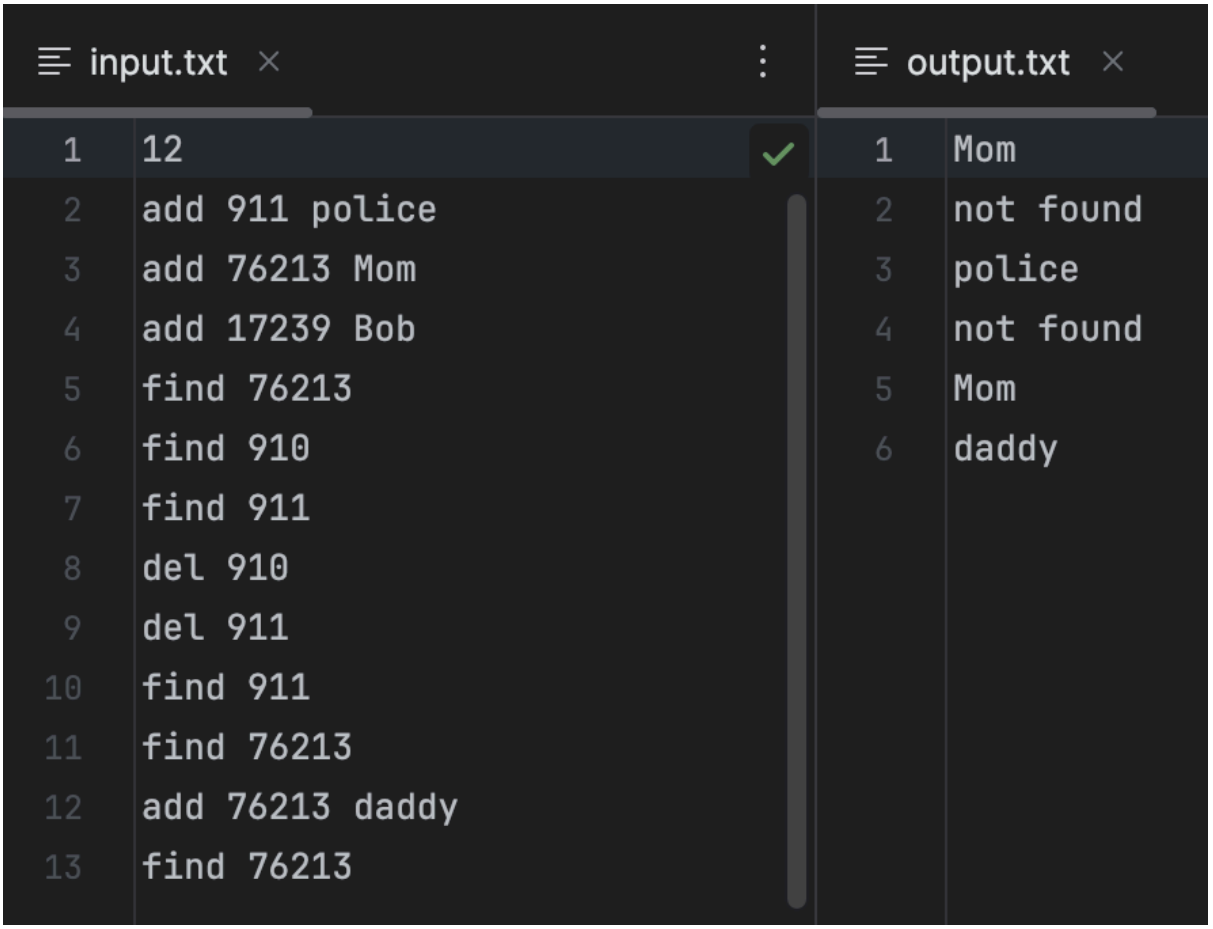
Класс для реализации работы телефонной книги, в котором присутствует 3 необходимых метода.

```
import tracemalloc
import time
from lab6.utils import *
from lab6.Task2.src.ClassPhoneBook import PhoneBook

t_start = time.perf_counter()
tracemalloc.start()


def main(commands):
    res = []
    phone_book = PhoneBook()
    for command in commands:
        if command.startswith("add"):
            _, number, name = command.split()
            phone_book.add_person(name, int(number))
        elif command.startswith("del"):
            _, number = command.split()
            phone_book.del_number(int(number))
        else:
            _, number = command.split()

res.append(phone_book.find_number(int(number)))
```

```
    return res


CURRENT_DIR =
os.path.dirname(os.path.abspath(__file__))
TXTF_DIR =
os.path.join(os.path.dirname(CURRENT_DIR), "txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")

if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    commands = [command.strip() for command in
lines[1:]]
    if 1 <= len(commands) <= 10 ** 5:
        result = main(commands)
        write_file('\n'.join(result), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" %
(time.perf_counter() - t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()
```

Основная функция, которая работает с файлами и обрабатывает команды, работая с классом.

Пример работы(скрин файлов)



	Время	Память
ПРИМЕР	0.000990708009339869 секунд	15247 байт

Вывод: была реализована работа с классом

Задача №7. Драгоценные камни

Текст задачи

В одной далекой восточной стране до сих пор по пустыням ходят караваны верблюдов, с помощью которых купцы привозят 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 п 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: (a1, 61), (a2, b2), ..., (ак, бк). Эти пары он называет красивыми, их множество мы обозначим как P. Теперь представим ряд камней, которые продают купцы, в виде строки S длины п из строчных букв латинского
алфавита. Шах считает число таких пар (i, j), что 1 ≤ i < j≤n, а камни Si u Sj
образуют красивую пару, то есть существует такое число 1 ≤ 9 ≤ k, 4T0 Si = aq
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.

Листинг кода

```
import tracemalloc
import time
from lab6.utils import *

t_start = time.perf_counter()
tracemalloc.start()
```

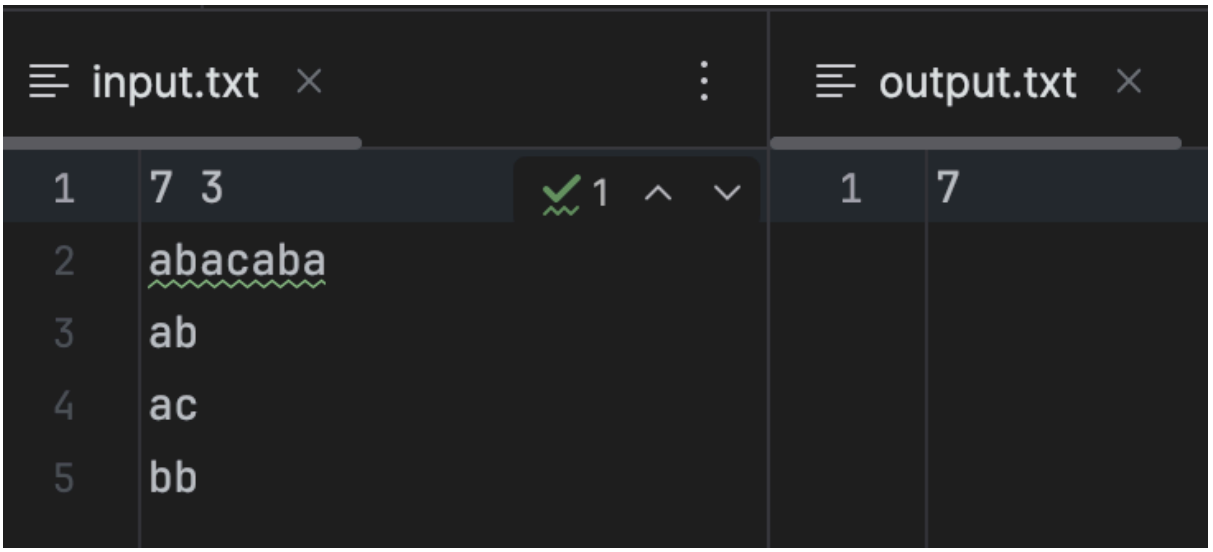
```python
def count_beautiful_pairs(n: int, S: str,
beautiful_pairs: list[tuple]) -> int:
    """
    функция для подсчета красивых пар
    :param n: int
    :param S: str
    :param beautiful_pairs: list[tuple]
    :return: int
    """
    beautiful_set = set(beautiful_pairs)
    freq = [0] * 26
    total_pairs = 0

    for i in range(n):
        current_char = S[i]
        for a, b in beautiful_set:
            if current_char == b:
                total_pairs += freq[ord(a) -
ord('a')]
        freq[ord(current_char) - ord('a')] += 1
    return total_pairs


CURRENT_DIR =
os.path.dirname(os.path.abspath(__file__))
TXTF_DIR =
os.path.join(os.path.dirname(CURRENT_DIR), "txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")

if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    n, k = lines[0].split()
    S = lines[1].strip()
    beautiful_pairs = [(i[0], i[1]) for i in
lines[2:]]
    if 1 <= int(n) <= 10 ** 5 and 1 <= int(k) <= 676:
```

```
        result = count_beautiful_pairs(int(n), S,
beautiful_pairs)
        write_file(str(result), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" %
(time.perf_counter() - t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()
```

Текстовое объяснение

Работа с файлами реализуется аналогично предыдущим задачам. В основной функции я с помощью цикла прохожу по строке, использую массив freq для подсчета частот появления символов в строке. Для каждого символа строки проверяю, сколько раз встречались символы, с которыми он образует красивую пару. Мы увеличиваем счётчик для каждого символа на каждой итерации.

Тесты(скрины файлов)



	время	память

тест 1	0.0106906669971067 46 секунд	14818 байт

Вывод: была реализована программа для подсчета красивых пар

Задача №8. Почти интерактивная хеш-таблица

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:

- $1 \le N \le 107$
- $1 \le \times \le 1015$
- $1 < A \le 103$
- $1 \le B \le 1015$

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 + By) mod 1015
- Установить X < (X • A + B) mod 1015

Начальные значения X, A и B, а также N, Ac, Bc, AD и Bд даны во входном файле. Выведите значения X, A и B после окончания работы.

Листинг кода

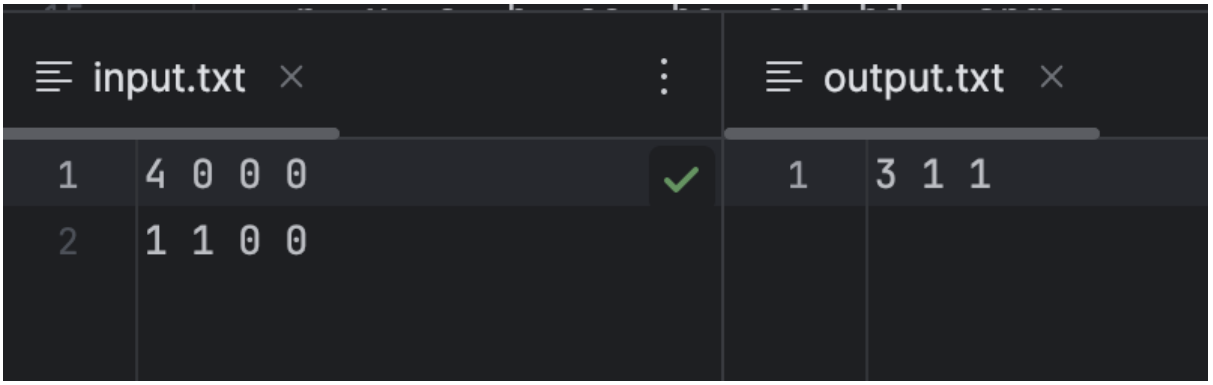
```
import tracemalloc
import time
from lab6.utils import *

t_start = time.perf_counter()
tracemalloc.start()


def solve(*args: tuple) -> tuple:
    """
    функция для работы с хеш-таблицей
    :param args: tuple
    :return: tuple
    """
    n, x, a, b, ac, bc, ad, bd = args
    table = set()
    for _ in range(n):
        if x in table:
            a = (a + ac) % 10 ** 3
            b = (b + bc) % 10 ** 15
        else:
            table.add(x)
            a = (a + ad) % 10 ** 3
            b = (b + bd) % 10 ** 15
        x = (x * a + b) % 10 ** 15
    return x, a, b


CURRENT_DIR =
os.path.dirname(os.path.abspath(__file__))
TXTF_DIR =
os.path.join(os.path.dirname(CURRENT_DIR), "txtf")
```

```
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")

if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    n, x, a, b = map(int, lines[0].split())
    ac, bc, ad, bd = map(int, lines[1].split())
    if 0 <= ac <= 10 ** 3 and 0 <= ad <= 10 ** 3 and
0 <= bc <= 10 ** 15 and 0 <= bd <= 10 ** 15:
        result = solve(n, x, a, b, ac, bc, ad, bd)
        write_file(' '.join(map(str, result)),
OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" %
(time.perf_counter() - t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()
```

Объяснение

Программа считывает данные, добавляет и удаляет элементы из общей таблицы, которую я реализую как множество. И в зависимости от присутствия элемента или его отсутствия в этой таблице выполняется различный перерасчет данных.

Скрины (файлов)



	время	память
тест 1	0.0009768330201040953 секунд	14610 байт

Вывод по всей работе: Я поработала с хеш-таблицами и познакомилась с хешированием