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

Выполнил:
Нгуен Хыу Жанг
К3140

Проверила:
Афанасьев А.В

Санкт-Петербург
2024 г

Содержание

Задачи по варианту

Задание 1 : Множество

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

- **Формат входного файла (input.txt).** В первой строке входного файла на- ходится строго положительное целое число операций N, не превышающее $5 \cdot 10^5$. В каждой из последующих N строк находится одна из следующих операций:

– A x – добавить элемент x в множество. Если элемент уже есть в мно- жестве, то ничего делать не надо.

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

– ? x – если ключ x есть в множестве, выведите «Y», если нет, то выведите «N».

Аргументы указанных выше операций – **целые числа**, не превышающие по модулю 10^{18}.

- **Формат выходного файла (output.txt).** Выведите последовательно резуль- тат выполнения всех операций «?». Следуйте формату выходного файла из примера.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
8 A 2 A 5 A 3 ? 2 ? 4 A 2 D 2 ? 2	Y N N

- Примечание.

Эту задачу можно решить совершенно разными способами, включая исполь- зование различных средств стандартных библиотек (правда, не всех - в стан- дартных библиотеках некоторых языков программирования используются слишком предсказуемые методы хеширования). Именно по этой причине ее разумно использовать для проверки реализаций хеш-таблиц, которые пона- добятся в следующих задачах этой работы.

```
import os

class SimpleSet:
    def __init__(self):
        self.data = set()

    def add(self, x):
        self.data.add(x)

    def remove(self, x):
```

```
        self.data.discard(x)

    def exists(self, x):
        return x in self.data

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        operations = [f.readline().strip() for _ in range(n)]
    return operations

def write_output(file_path, results):
    with open(file_path, 'w') as f:
        f.write('\n'.join(results) + '\n')

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    operations = read_input(input_path)
    simple_set = SimpleSet()
    results = []

    for operation in operations:
        parts = operation.split()
        cmd = parts[0]
        if cmd == 'A':
            x = int(parts[1])
            simple_set.add(x)
        elif cmd == 'D':
            x = int(parts[1])
            simple_set.remove(x)
        elif cmd == '?':
            x = int(parts[1])
            if simple_set.exists(x):
                results.append('Y')
            else:
                results.append('N')

    write_output(output_path, results)

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

input.txt:

```
8
A 2
A 5
A 3
? 2
? 4
A 2
D 2
? 2
```

output.txt:

```
Y
N
N
```

1. Класс `SimpleSet`

Этот класс реализует базовую структуру данных — множество, с помощью которой выполняются все операции. В Python для хранения множества используется встроенный тип данных `set`.

1. **`__init__`:**
   ```
   def __init__(self):
       self.data = set()
   ```
   - Инициализация нового объекта класса `SimpleSet`, в котором создается пустое множество `self.data`.
2. **`add`:**
   ```
   def add(self, x):
       self.data.add(x)
   ```
   - Метод добавляет элемент `x` в множество.
   - Если элемент уже есть, ничего не произойдет, так как множество не допускает дублирования.
3. **`remove`:**
   ```
    def remove(self, x):
       self.data.discard(x)
   ```
   - Метод удаляет элемент `x` из множества.
   - Используется метод `discard`, чтобы избежать ошибки, если элемент отсутствует.
4. **`exists`:**
   ```
   def exists(self, x):
       return x in self.data
   ```
   - Проверяет наличие элемента `x` в множестве.
   - Возвращает `True`, если элемент есть, иначе — `False`.

2. Функции ввода и вывода

1. **`read_input`:**
   ```
   def read_input(file_path):
       with open(file_path, 'r') as f:
           n = int(f.readline().strip())
           operations = [f.readline().strip() for _ in
   range(n)]
       return operations
   ```
   - Считывает данные из входного файла `file_path`.
   - Первая строка файла — это число операций `n`.
   - Далее считываются строки операций (например, `A x`, `D x`, `? x`), которые сохраняются в список `operations`.
2. **`write_output`:**
   ```
   def write_output(file_path, results):
       with open(file_path, 'w') as f:
           f.write('\n'.join(results) + '\n')
   ```

- Записывает результаты выполнения операций `? x` в выходной файл `file_path`.
- Каждый результат (`Y` или `N`) записывается с новой строки.

3. Функция `main`

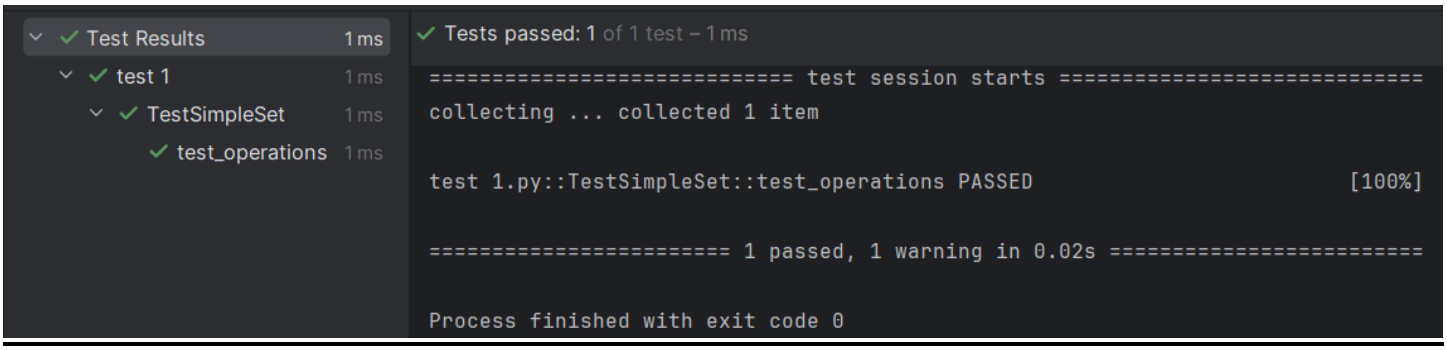
Основная логика программы:

1. **Чтение входных данных:**
```
input_path = os.path.join('..', 'txtf', 'input.txt')
output_path = os.path.join('..', 'txtf', 'output.txt')

operations = read_input(input_path)
```
   - Задаются пути к входному и выходному файлам.
   - Считываются операции из файла `input.txt`.
2. **Инициализация множества и обработка операций:**
```
simple_set = SimpleSet()
results = []

for operation in operations:
    parts = operation.split()
    cmd = parts[0]
    if cmd == 'A':
        x = int(parts[1])
        simple_set.add(x)
    elif cmd == 'D':
        x = int(parts[1])
        simple_set.remove(x)
    elif cmd == '?':
        x = int(parts[1])
        if simple_set.exists(x):
            results.append('Y')
        else:
            results.append('N')
```
   - Создается объект класса `SimpleSet`.
   - Итерация по всем операциям:
     - Если команда `A x`, вызывается метод `add(x)`.
     - Если команда `D x`, вызывается метод `remove(x)`.
     - Если команда `? x`, проверяется наличие элемента `x` и добавляется результат (`Y` или `N`) в список `results`.
3. **Запись результатов:**
```
write_output(output_path, results)
```
   - Результаты операций `? x` записываются в файл `output.txt`.

<u>Unittest для задание 1:</u>

```python
import unittest
import os
import sys

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from d1 import SimpleSet, read_input, write_output

class TestSimpleSet(unittest.TestCase):

    def test_operations(self):
        # given
        input_data = "8\nA 2\nA 5\nA 3\n? 2\n? 4\nA 2\nD 2\n? 2\n"
        expected_output = "Y\nN\nN\n"

        # when
        with open(os.path.join('..', 'txtf', 'input.txt'), 'w') as f:
            f.write(input_data)

        # when
        operations = read_input(os.path.join('..', 'txtf', 'input.txt'))

        # when
        simple_set = SimpleSet()
        results = []

        # when
        for operation in operations:
            parts = operation.split()
            cmd = parts[0]
            if cmd == 'A':
                x = int(parts[1])
                simple_set.add(x)
            elif cmd == 'D':
                x = int(parts[1])
                simple_set.remove(x)
            elif cmd == '?':
                x = int(parts[1])
                if simple_set.exists(x):
                    results.append('Y')
                else:
                    results.append('N')

        # when
        write_output(os.path.join('..', 'txtf', 'output.txt'), results)

        # when
        with open(os.path.join('..', 'txtf', 'output.txt'), 'r') as f:
            result = f.read()

        # then
        self.assertEqual(result, expected_output)

if __name__ == "__main__":
    unittest.main()
```

*** Результат :**



1. Импорты и настройка окружения

```
import unittest
import os
import sys

sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src')))
from d1 import SimpleSet, read_input, write_output
```

1. **`unittest`**: стандартный модуль Python для создания и выполнения тестов.
2. **`os` и `sys`**: используются для настройки путей к файлам и модулям.
    - `sys.path.insert(...)`: добавляет путь к директории `src`, чтобы импортировать модуль `d1`, где находятся `SimpleSet`, `read_input` и `write_output`.

2. Класс `TestSimpleSet`

Класс теста наследует `unittest.TestCase`, что делает его тестовым модулем.

Метод test_operations

Основной метод, который проверяет выполнение операций над множеством. Рассмотрим его шаги:

1. Подготовка входных данных

```
input_data = "8\nA 2\nA 5\nA 3\n? 2\n? 4\nA 2\nD 2\n? 2\n"
expected_output = "Y\nN\nN\n"
```

- **`input_data`**: строки, имитирующие содержимое входного файла.
    - 8 — количество операций.
    - Следующие строки — команды (`A x`, `D x`, `? x`).
- **`expected_output`**: ожидаемый результат операций `? x`.

2. Запись данных в файл input.txt

```
with open(os.path.join('..', 'txtf', 'input.txt'), 'w') as f:
    f.write(input_data)
```

- Открывается файл `input.txt` в режиме записи ('w'), и туда записывается подготовленный `input_data`.

3. Чтение операций из файла

```
operations = read_input(os.path.join('..', 'txtf', 'input.txt'))
```

- Функция `read_input` считывает данные из `input.txt` и возвращает список операций.

4. Инициализация множества

```
simple_set = SimpleSet()
results = []
```

- Создается объект `simple_set` — экземпляр класса `SimpleSet`.
- Список `results` будет хранить результаты выполнения операций `? x`.

5. Выполнение операций

```
for operation in operations:
    parts = operation.split()
    cmd = parts[0]
    if cmd == 'A':
        x = int(parts[1])
        simple_set.add(x)
    elif cmd == 'D':
        x = int(parts[1])
        simple_set.remove(x)
    elif cmd == '?':
        x = int(parts[1])
        if simple_set.exists(x):
            results.append('Y')
        else:
            results.append('N')
```

- Каждая операция разбирается на части:
  - `cmd` — команда (`A`, `D`, `?`).
  - `x` — аргумент команды (число).
- В зависимости от команды вызываются методы `add(x)`, `remove(x)` или `exists(x)` у объекта `simple_set`:
  - Если `? x` — результат (`Y` или `N`) добавляется в `results`.

6. Запись результатов в файл output.txt

```
write_output(os.path.join('..', 'txtf', 'output.txt'), results)
```

- Функция `write_output` записывает результаты операций `? x` из `results` в файл `output.txt`.

7. Чтение и проверка результата

```
with open(os.path.join('..', 'txtf', 'output.txt'), 'r') as f:
    result = f.read()

self.assertEqual(result, expected_output)
```

- Файл `output.txt` читается, и его содержимое сравнивается с `expected_output` с помощью метода `assertEqual`.

3. Запуск тестов

```
if __name__ == "__main__":
    unittest.main()
```

- Запускает выполнение тестов, если файл запускается напрямую.

Задание 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 дол- жен перезаписать соответствующее имя.
- `del number` – означает, что менеджер должен удалить человека с номе- ром из телефонной книги. Если такого человека нет, то он должен просто игнорировать запрос.
- `find number` – означает, что пользователь ищет человека с номером те- лефона `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 ($1 \le N \le 10^5$) - 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 каж- дого поискового запроса `find` – имя, соответствующее номеру телефона, или «not found» (без кавычек), если в телефонной книге нет человека с та- ким номером телефона. Выведите по одному результату в каждой строке в том же порядке, как были заданы запросы типа `find` во входных данных.
- Ограничение по времени. 6 сек.
- Ограничение по памяти. 512 мб.
- Примеры:

input.txt
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.txt
Mom not found police not found Mom daddy

input.txt
8 find 3839442 add 123456 me add 0 granny find 0 find 123456 del 0 del 0 find 0
output.txt
not found granny me not found

Описание примера 1. 76213 - это номер Mom, 910 - нет в телефонной книге, 911 - это номер police, но затем он был удален из телефонной книги, поэтому второй поиск 911 вернул «not found». Также обратите внимание, что когда daddy был добавлен с тем же номером телефона 76213, что и номер телефона Mom, имя контакта было переписано, и теперь поиск 76213 возвращает «daddy» вместо
«Mom».

```
import os

class PhoneBook:
    def __init__(self):
        self.contacts = {}

    def add(self, number, name):
        self.contacts[number] = name

    def delete(self, number):
        if number in self.contacts:
            del self.contacts[number]

    def find(self, number):
        return self.contacts.get(number, "not found")

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        operations = [f.readline().strip() for _ in range(n)]
    return operations

def write_output(file_path, results):
    with open(file_path, 'w') as f:
        f.write('\n'.join(results) + '\n')

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    operations = read_input(input_path)
    phone_book = PhoneBook()
    results = []

    for operation in operations:
        parts = operation.split()
        cmd = parts[0]
        if cmd == 'add':
            number = parts[1]
            name = parts[2]
            phone_book.add(number, name)
        elif cmd == 'del':
            number = parts[1]
            phone_book.delete(number)
        elif cmd == 'find':
            number = parts[1]
            result = phone_book.find(number)
            results.append(result)

    write_output(output_path, results)

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

input.txt:

```
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.txt:

```
Mom
not found
police
not found
Mom
daddy
```

1. Класс `PhoneBook`

Этот класс содержит базовую функциональность телефонной книги. Внутри него используется словарь (`dict`) для хранения данных, где ключом является номер телефона, а значением — имя.

1. **Инициализация**
   ```
   def __init__(self):
       self.contacts = {}
   ```
   - Создается пустой словарь `contacts`, в котором будут храниться записи телефонной книги.
2. **Добавление записи**
   ```
   def add(self, number, name):
       self.contacts[number] = name
   ```
   - Метод добавляет новую запись или обновляет имя для уже существующего номера.
   - Ключ — номер телефона (`number`), значение — имя (`name`).
3. **Удаление записи**
   ```
   def delete(self, number):
       if number in self.contacts:
           del self.contacts[number]
   ```
   - Метод удаляет запись с номером `number`, если он есть в телефонной книге.

- Проверка `if number in self.contacts` предотвращает ошибку, если номер отсутствует.

4. **Поиск записи**

```
def find(self, number):
    return self.contacts.get(number, "not found")
```

- Метод возвращает имя, соответствующее номеру `number`.
- Если номера нет в телефонной книге, возвращается строка `"not found"`.

2. Функции ввода и вывода

1. **Чтение входных данных**

```
def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        operations = [f.readline().strip() for _ in
range(n)]
    return operations
```

- Открывает входной файл `file_path`.
- Первая строка содержит число запросов `n`.
- Далее считываются строки операций (`add`, `del`, `find`), которые возвращаются в виде списка.

2. **Запись результатов в файл**

```
def write_output(file_path, results):
    with open(file_path, 'w') as f:
        f.write('\n'.join(results) + '\n')
```

- Открывает выходной файл `file_path` в режиме записи.
- Записывает результаты всех запросов типа `find` (поиск) построчно.

3. Основная функция `main`

1. **Пути к файлам**

```
input_path = os.path.join('..', 'txtf', 'input.txt')
output_path = os.path.join('..', 'txtf', 'output.txt')
```

- Указываются пути к входному и выходному файлам.

2. **Чтение данных и создание телефонной книги**

```
operations = read_input(input_path)
phone_book = PhoneBook()
results = []
```

- Операции считываются из входного файла.
- Создается объект `phone_book` — экземпляр класса `PhoneBook`.
- `results` будет содержать результаты запросов типа `find`.

3. **Обработка операций**

```
for operation in operations:
    parts = operation.split()
    cmd = parts[0]
    if cmd == 'add':
        number = parts[1]
        name = parts[2]
```

```
            phone_book.add(number, name)
        elif cmd == 'del':
            number = parts[1]
            phone_book.delete(number)
        elif cmd == 'find':
            number = parts[1]
            result = phone_book.find(number)
            results.append(result)
```

- Цикл обрабатывает каждую операцию:
  - **`add number name`**: добавляет запись в телефонную книгу.
  - **`del number`**: удаляет запись по номеру телефона.
  - **`find number`**: выполняет поиск и добавляет результат (`имя` или `"not found"`) в список `results`.

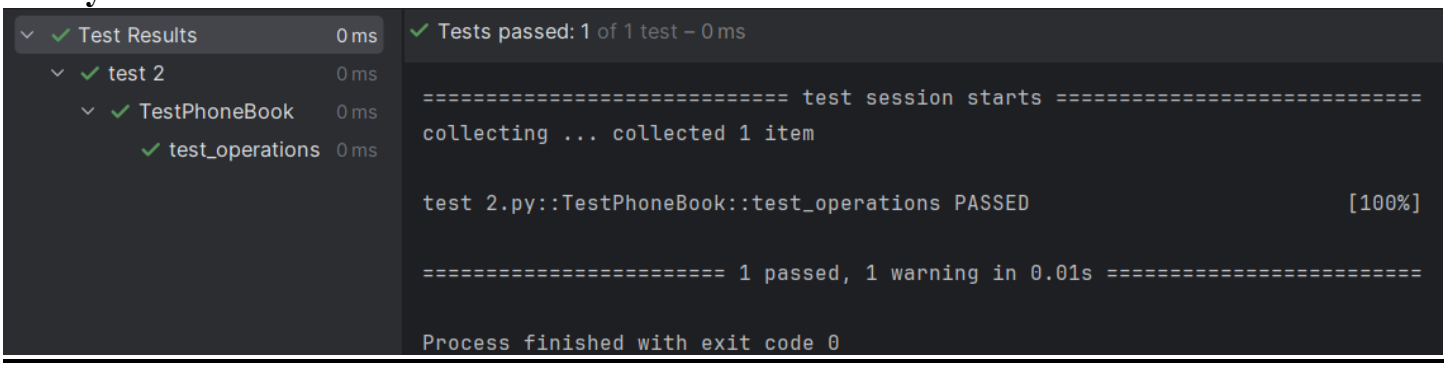
4. **Запись результатов**
   `write_output(output_path, results)`
   - Результаты запросов `find` записываются в выходной файл.

Unittest для задание 2:

```python
import unittest
import sys
import os

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from d2 import PhoneBook

class TestPhoneBook(unittest.TestCase):

    def test_operations(self):
        # given
        operations = [
            "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"
        ]
        expected_output = [
            "Mom",
            "not found",
            "police",
            "not found",
            "Mom",
            "daddy"
        ]

        # when
        phone_book = PhoneBook()
```

```
        results = []

        # when
        for operation in operations:
            parts = operation.split()
            cmd = parts[0]
            if cmd == 'add':
                number = parts[1]
                name = parts[2]
                phone_book.add(number, name)
            elif cmd == 'del':
                number = parts[1]
                phone_book.delete(number)
            elif cmd == 'find':
                number = parts[1]
                result = phone_book.find(number)
                results.append(result)

        # then
        self.assertEqual(results, expected_output)

if __name__ == "__main__":
    unittest.main()
```

*** Результат :**



Импортируемые модули

1. **`unittest`**:
   - Это стандартный модуль Python для написания и выполнения модульных тестов.
2. **`sys` и `os`**:
   - Эти модули используются для корректного импорта тестируемого модуля PhoneBook из директории src.
3. **`from d2 import PhoneBook`**:
   - Импортируется класс PhoneBook из файла d2.py, который реализует телефонную книгу.

Класс `TestPhoneBook`

Этот класс наследуется от `unittest.TestCase` и содержит тестовые методы для проверки работы телефонной книги.

Метод `test_operations`

1. Подготовка данных

```
operations = [
    "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"
]
expected_output = [
    "Mom",
    "not found",
    "police",
    "not found",
    "Mom",
    "daddy"
]
```

- **`operations`**: Это список команд, которые будут выполняться в телефонной книге.
- **`expected_output`**: Это ожидаемые результаты команд find.

2. Создание экземпляра PhoneBook

```
phone_book = PhoneBook()
results = []
```

- Создается объект phone_book, который используется для выполнения команд.
- Список results будет содержать результаты выполнения операций find.

3. Обработка операций

```
for operation in operations:
    parts = operation.split()
    cmd = parts[0]
    if cmd == 'add':
        number = parts[1]
        name = parts[2]
        phone_book.add(number, name)
    elif cmd == 'del':
        number = parts[1]
        phone_book.delete(number)
    elif cmd == 'find':
        number = parts[1]
        result = phone_book.find(number)
        results.append(result)
```

- Цикл обрабатывает каждую строку из `operations`:
  - **`add`**: добавляет запись в телефонную книгу.

```
phone_book.add(number, name)
```

- **del**: удаляет запись с указанным номером телефона.Type equation here.
```
phone_book.delete(number)
```
- **find**: выполняет поиск номера телефона и добавляет результат (имя или "not found") в список results.
```
result = phone_book.find(number)
results.append(result)
```

4. Сравнение результатов

```
self.assertEqual(results, expected_output)
```

- Метод assertEqual сравнивает список results с expected_output.
- Если они совпадают, тест считается пройденным.

Задание 3 : Хеширование с цепочками

В этой задаче вы реализуете хеш-таблицу, используя схему цепочки. Хеширование с цепочками - один из самых популярных способов реализации хеш-таблиц на практике. Хеш-таблицу, которую вы создадите, можно использовать для реализации телефонной книги на вашем телефоне или для хранения таблицы паролей вашего компьютера или веб-службы (но не забывайте хранить хэши паролей вместо самих паролей, иначе вас могут взломать!).

В этой задаче ваша цель - реализовать хеш-таблицу с цепочкой списков. Вам дано количество сегментов (карманов) m и хеш-функция. Это полиномиальная хеш-функция :

$$h(S) = ((\sum_{i=0}^{|S|-1} S[i]x^i) \bmod p) \bmod m,$$

в которой S[i] - код ASCII i-го символа строки S, p = 1000000007 и x = 263. Ваш алгоритм должен поддерживать следующие типы запросов:

- add string – вставить строку string в таблицу. Если такая строка уже есть в хеш-таблице, то просто игнорируйте запрос.
- del string – удалить строку string из таблицы. Если такой строки нет в хэш-таблице, тогда просто игнорируйте запрос.
- find string – выведите «yes» или «no» (без кавычек) в зависимости от того, содержит ли таблица строку или нет.
- check *i* – вывести содержимое *i*-го списка в таблицу. Используйте пробелы для разделения элементов списка. **Если *i*-й список пуст, вывести пустую строку.**

  При вставке новой строки в цепочку вы должны вставить ее в начало цепочки.

- **Формат ввода / входного файла (input.txt).** В первой строке находится единственное целое число m - количество сегментов хэш-таблицы. Следу- ющая строка содержит число запросов N , после которой идут еще N строк, каждая содержит один запрос в формате, заданном выше. Ограничения: $1 \le N \le 10^5$, $N/5 \le m \le N$. Все строки состоят из латинских букв. Каждая из них не пустая и имеет длину не более 15 символов.

- **Формат вывода / выходного файла (output.txt).** Выведите результат каж- дого запроса `find` и `check`, по одному результату в строке, в том же порядке, в каком эти запросы указаны во входных данных.
- Ограничение по времени. 7 сек.
- Ограничение по памяти. 512 мб.
- Примеры:

input.txt
5 12 add world add HellO check 4 find World find world del world check 4 del HellO add luck add GooD check 2 del good
output.txt
HellO world no yes HellO GooD luck

input.txt
4 8 add test add test find test del test find test find Test add Test find Test
output.txt
yes no no yes

input.txt
3 12 check 0 find help add help add del add add find add find del del del find del check 0 check 1 check 2
output.txt
no yes yes no add help

- Описание примера №1. Код ASCII для «w» - 119, для «o» - 111, для «r» - 114, для «l» - 108, а для «d» - 100. Таким образом, h(«world») = $(119 + 111 \cdot 263 + 114 \cdot 263^2 + 108 \cdot 263^3 + 100 \cdot 263^4 \bmod 1000000007) \bmod 5 = 4$.
  Также хеш-значение «HellO» равно 4. Напомним, что мы всегда вставляем в начало цепочки, поэтому после добавления «world», а затем «HellO» в ту же цепочку с номером 4 сначала идет «HellO», а затем «world».

  Далее, строка «World» не найдена, а строка «world» найдена, потому что строки чувствительны к регистру, а коды «W» и «w» различны. После уда- ления «world» в цепочке 4 будет найдено только «HellO». Наконец после добавления «luck» и «GooD» к одной цепочке 2 сначала идет «GooD», а затем «luck».

```
import os

class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [[] for _ in range(size)]
        self.p = 1000000007
        self.x = 263

    def hash_function(self, s):
```

```
        h = 0
        for i, char in enumerate(s):
            h = (h + (ord(char) * (self.x ** i)) % self.p) % self.p
        return h % self.size

    def add(self, string):
        index = self.hash_function(string)
        if string not in self.table[index]:
            self.table[index].insert(0, string)

    def delete(self, string):
        index = self.hash_function(string)
        if string in self.table[index]:
            self.table[index].remove(string)

    def find(self, string):
        index = self.hash_function(string)
        return "yes" if string in self.table[index] else "no"

    def check(self, index):
        return " ".join(self.table[index]) if self.table[index] else ""

def read_input(file_path):
    with open(file_path, 'r') as f:
        m = int(f.readline().strip())
        n = int(f.readline().strip())
        operations = [f.readline().strip() for _ in range(n)]
    return m, operations

def write_output(file_path, results):
    with open(file_path, 'w') as f:
        f.write('\n'.join(results) + '\n')

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    m, operations = read_input(input_path)
    hash_table = HashTable(m)
    results = []

    for operation in operations:
        parts = operation.split()
        cmd = parts[0]
        if cmd == 'add':
            string = parts[1]
            hash_table.add(string)
        elif cmd == 'del':
            string = parts[1]
            hash_table.delete(string)
        elif cmd == 'find':
            string = parts[1]
            results.append(hash_table.find(string))
        elif cmd == 'check':
            index = int(parts[1])
            results.append(hash_table.check(index))

    write_output(output_path, results)

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

input.txt:

```
5
12
add world
add HellO
check 4
find World
find world
del world
check 4
del HellO
add luck
add GooD
check 2
del good
```

output.txt:

```
HellO world
no
yes
HellO
GooD luck
```

1. Класс `HashTable`

Этот класс реализует функциональность хеш-таблицы с цепочками.

Инициализация (__init__)

```
def __init__(self, size):
    self.size = size
    self.table = [[] for _ in range(size)]
    self.p = 1000000007
    self.x = 263
```

- **`size`**: количество сегментов (карманов) таблицы.
- **`table`**: список списков, представляющий сегменты таблицы.
- **`p`**: большое простое число для модуля хеш-функции.
- **`x`**: коэффициент для вычисления хеша.

Хеш-функция (hash_function)

```
def hash_function(self, s):
    h = 0
    for i, char in enumerate(s):
        h = (h + (ord(char) * (self.x ** i)) % self.p) % self.p
    return h % self.size
```

- **Вход**: строка s.

- **Выход**: номер сегмента (кармана) для строки.

Добавление строки (add)

```
def add(self, string):
    index = self.hash_function(string)
    if string not in self.table[index]:
        self.table[index].insert(0, string)
```

- **Процесс**:
  - Вычисляется хеш строки.
  - Если строка отсутствует в соответствующем сегменте, она добавляется в начало.

Удаление строки (delete)

```
def delete(self, string):
    index = self.hash_function(string)
    if string in self.table[index]:
        self.table[index].remove(string)
```

- **Процесс**:
  - Вычисляется хеш строки.
  - Если строка найдена в соответствующем сегменте, она удаляется.

Поиск строки (find)

```
def find(self, string):
    index = self.hash_function(string)
    return "yes" if string in self.table[index] else "no"
```

- **Процесс**:
  - Вычисляется хеш строки.
  - Если строка присутствует в сегменте, возвращается "yes", иначе "no".

Проверка содержимого сегмента (check)

```
def check(self, index):
    return " ".join(self.table[index]) if self.table[index] else
""
```

- **Процесс**:
  - Возвращает строки в указанном сегменте, разделённые пробелами.
  - Если сегмент пустой, возвращается пустая строка.

2. Чтение данных (`read_input`)

```
def read_input(file_path):
    with open(file_path, 'r') as f:
        m = int(f.readline().strip())
        n = int(f.readline().strip())
        operations = [f.readline().strip() for _ in range(n)]
    return m, operations
```

- **Вход**: путь к файлу.
- **Выход**: количество сегментов m и список операций operations.

3. Запись результата (`write_output`)

```
def write_output(file_path, results):
    with open(file_path, 'w') as f:
```

```
        f.write('\n'.join(results) + '\n')
```

- **Вход**: результаты операций и путь к файлу.
- **Процесс**: результаты записываются в файл, каждый результат на новой строке.

Основная функция (`main`)

```
def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    m, operations = read_input(input_path)
    hash_table = HashTable(m)
    results = []

    for operation in operations:
        parts = operation.split()
        cmd = parts[0]
        if cmd == 'add':
            string = parts[1]
            hash_table.add(string)
        elif cmd == 'del':
            string = parts[1]
            hash_table.delete(string)
        elif cmd == 'find':
            string = parts[1]
            results.append(hash_table.find(string))
        elif cmd == 'check':
            index = int(parts[1])
            results.append(hash_table.check(index))

    write_output(output_path, results)
```

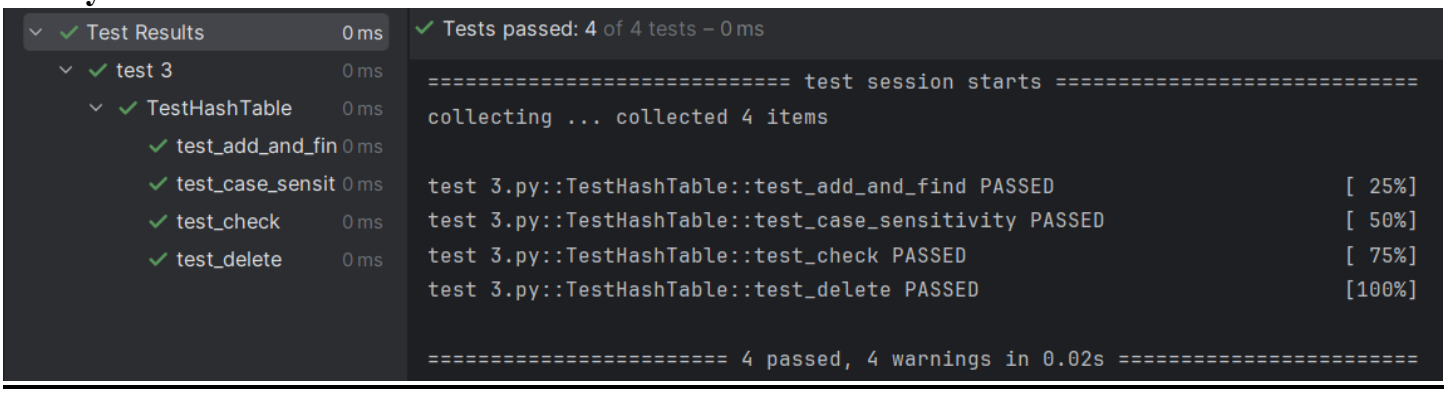
1. **Чтение данных**:
   - Из файла input.txt считываются количество сегментов и операции.
2. **Создание хеш-таблицы**:
   - Объект hash_table создаётся с заданным количеством сегментов.
3. **Обработка операций**:
   - Выполняются команды add, del, find, check.
   - Результаты find и check сохраняются в results.
4. **Запись результатов**:
   - Результаты записываются в файл output.txt.

Unittest для задание 3:

```
import os
import sys
import unittest

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
```

```python
from d3 import HashTable

class TestHashTable(unittest.TestCase):

    def setUp(self):
        # given
        self.hash_table = HashTable(5)

    def test_add_and_find(self):
        # given
        self.hash_table.add("test")
        # when
        self.assertEqual(self.hash_table.find("test"), "yes")
        # then
        self.assertEqual(self.hash_table.find("not_in_table"), "no")

    def test_delete(self):
        # given
        self.hash_table.add("test")
        # when
        self.hash_table.delete("test")
        # then
        self.assertEqual(self.hash_table.find("test"), "no")

    def test_check(self):
        # given
        self.hash_table.add("hello")
        # when
        self.hash_table.add("world")
        # then
        self.assertEqual(self.hash_table.check(0), "hello")

    def test_case_sensitivity(self):
        # given
        self.hash_table.add("Test")
        # when
        self.assertEqual(self.hash_table.find("test"), "no")
        # then
        self.assertEqual(self.hash_table.find("Test"), "yes")

if __name__ == "__main__":
    unittest.main()
```

*** Результат :**



1. Импорт библиотек и подготовка

```
import os
import sys
import unittest
```

```
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src')))
from d3 import HashTable
```

- **`unittest`**: библиотека для модульного тестирования.
- **`sys.path.insert`**: добавляет путь к модулю HashTable (файл d3.py в папке src) для импорта.
- **`from d3 import HashTable`**: импорт класса HashTable из тестируемого модуля.

2. Класс тестов `TestHashTable`

Этот класс наследует `unittest.TestCase` и содержит методы для тестирования разных функций хеш-таблицы.

Метод setUp

```
def setUp(self):
    self.hash_table = HashTable(5)
```

- Выполняется перед каждым тестом.
- Создаёт экземпляр HashTable с размером 5 сегментов.
- Это позволяет изолировать тесты и работать с чистой таблицей.

3. Тесты

1. Тест добавления и поиска строки (test_add_and_find)

```
def test_add_and_find(self):
    self.hash_table.add("test")
    self.assertEqual(self.hash_table.find("test"), "yes")
    self.assertEqual(self.hash_table.find("not_in_table"), "no")
```

- **Процесс**:
    1. Добавляется строка "test".
    2. Проверяется наличие строки "test" с помощью find. Ожидается "yes".
    3. Проверяется отсутствие строки "not_in_table". Ожидается "no".

2. Тест удаления строки (test_delete)

```
def test_delete(self):
    self.hash_table.add("test")
    self.hash_table.delete("test")
    self.assertEqual(self.hash_table.find("test"), "no")
```

- **Процесс**:
    1. Добавляется строка "test".
    2. Удаляется строка "test".
    3. Проверяется отсутствие строки "test". Ожидается "no".

3. Тест проверки содержимого сегмента (test_check)

```
def test_check(self):
    self.hash_table.add("hello")
    self.hash_table.add("world")
    self.assertEqual(self.hash_table.check(0), "hello")
```

- **Процесс**:

1. Добавляются строки "hello" и "world".
2. Проверяется содержимое первого сегмента (с индексом 0).
3. Ожидается, что первой строкой в сегменте будет "hello" (вставляется в начало списка).

4. Тест чувствительности к регистру (test_case_sensitivity)

```
def test_case_sensitivity(self):
    self.hash_table.add("Test")
    self.assertEqual(self.hash_table.find("test"), "no")
    self.assertEqual(self.hash_table.find("Test"), "yes")
```

- **Процесс**:
  1. Добавляется строка "Test".
  2. Проверяется отсутствие строки "test" (строка чувствительна к регистру). Ожидается "no".
  3. Проверяется наличие строки "Test". Ожидается "yes".

4. Запуск тестов

```
if __name__ == "__main__":
    unittest.main()
```

- Выполняет все тесты в классе `TestHashTable`.

Задание 4 : Прошитый ассоциативный массив

Реализуйте прошитый ассоциативный массив. Ваш алгоритм должен поддер- живать следующие типы операций:

- `get` *x* – если ключ *x* есть в множестве, выведите соответствующее ему значение, если нет, то выведите <none>.
- `prev` *x* – вывести значение, соответствующее ключу, находящемуся в ассо- циативном массиве, который был вставлен позже всех, но до *x*, или <none>, если такого нет или в массиве нет *x*.
- `next` *x* – вывести значение, соответствующее ключу, находящемуся в ассо- циативном массиве, который был вставлен раньше всех, но после *x* , или <none>, если такого нет или в массиве нет *x* .
- `put` *x y* – поставить в соответствие ключу *x* значение *y*. При этом следует учесть, что
  - если, независимо от предыстории, этого ключа на момент вставки в массиве не было, то он считается только что вставленным и оказыва- ется самым последним среди добавленных элементов – то есть, вызов `next` с этим же ключом сразу после выполнения текущей операции put должен вернуть <none>;
  - если этот ключ уже есть в массиве, то значение необходимо изменить, и в этом случае ключ не считается вставленным еще раз, то есть, не меняет своего положения в порядке добавленных элементов.
- `delete` *x* – удалить ключ *x*. Если ключа в ассоциативном массиве нет, то ничего делать не надо.
- **Формат входного файла (input.txt).** В первой строке входного файла на- ходится строго

положительное целое число операций N, не превышающее $5 \cdot 10^5$. В каждой из последующих N строк находится одна из приведенных выше операций. Ключи и значения операций - строки из латинских букв длиной не менее одного и не более 20 символов.

- **Формат выходного файла (output.txt).** Выведите последовательно резуль- тат выполнения всех операций get, prev, next. Следуйте формату выходного файла из примера.
- Ограничение по времени. 4 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt	output.txt
14 put zero a put one b put two c put three d put four e get two prev two next two delete one delete three get two prev two next two next four	c b d c a e <none>

- P.s. Задача на openedu, 8 неделя.

```
import os

class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None
        self.prev = None


class AssociativeArray:
    def __init__(self):
        self.data = {}
        self.head = None
        self.tail = None

    def put(self, key, value):
        if key in self.data:
            self.data[key].value = value
        else:
            new_node = Node(key, value)
            self.data[key] = new_node

            if self.tail:
                self.tail.next = new_node
                new_node.prev = self.tail
            else:
                self.head = new_node
```

```
            self.tail = new_node

    def get(self, key):
        if key in self.data:
            return self.data[key].value
        return "<none>"

    def delete(self, key):
        if key in self.data:
            node_to_delete = self.data[key]
            if node_to_delete.prev:
                node_to_delete.prev.next = node_to_delete.next
            if node_to_delete.next:
                node_to_delete.next.prev = node_to_delete.prev
            if node_to_delete == self.head:
                self.head = node_to_delete.next
            if node_to_delete == self.tail:
                self.tail = node_to_delete.prev

            del self.data[key]

    def prev(self, key):
        if key in self.data:
            node = self.data[key].prev
            if node:
                return node.value
        return "<none>"

    def next(self, key):
        if key in self.data:
            node = self.data[key].next
            if node:
                return node.value
        return "<none>"

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        operations = [f.readline().strip() for _ in range(n)]
    return operations

def write_output(file_path, results):
    with open(file_path, 'w') as f:
        f.write('\n'.join(results) + '\n')

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    operations = read_input(input_path)
    associative_array = AssociativeArray()
    results = []

    for operation in operations:
        parts = operation.split()
        cmd = parts[0]
        if cmd == 'put':
            associative_array.put(parts[1], parts[2])
        elif cmd == 'get':
            results.append(associative_array.get(parts[1]))
        elif cmd == 'prev':
            results.append(associative_array.prev(parts[1]))
```

```
        elif cmd == 'next':
            results.append(associative_array.next(parts[1]))
        elif cmd == 'delete':
            associative_array.delete(parts[1])

    write_output(output_path, results)

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

input.txt:

```
14
put zero a
put one b
put two c
put three d
put four e
get two
prev two
next two
delete one
delete three
get two
prev two
next two
next four
```

output.txt:

```
c
b
d
c
a
e
<none>
```

1. Класс Node

```
class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None
        self.prev = None
```

- **Назначение**: Узел двусвязного списка.
- **Атрибуты**:
  - key: Ключ элемента.

- value: Значение элемента.
- next: Указатель на следующий узел.
- prev: Указатель на предыдущий узел.

2. Класс AssociativeArray

Этот класс реализует ассоциативный массив с использованием хранилища данных и двусвязного списка.

Инициализация

```
class AssociativeArray:
    def __init__(self):
        self.data = {}
        self.head = None
        self.tail = None
```

- **`self.data`**: Словарь, где ключ - это ключ элемента, а значение - объект типа `Node`.
- **`self.head` и `self.tail`**: Указатели на начало и конец двусвязного списка.

Метод put

```
def put(self, key, value):
    if key in self.data:
        self.data[key].value = value
    else:
        new_node = Node(key, value)
        self.data[key] = new_node

        if self.tail:
            self.tail.next = new_node
            new_node.prev = self.tail
        else:
            self.head = new_node

        self.tail = new_node
```

- Если ключ уже существует, обновляется его значение.
- Если ключ отсутствует:
  - Создаётся новый узел.
  - Узел добавляется в конец двусвязного списка.
  - Указатель tail обновляется.

Метод get

```
def get(self, key):
    if key in self.data:
        return self.data[key].value
    return "<none>"
```

- Проверяет наличие ключа в словаре data.
- Возвращает значение ключа, если он существует, иначе `"<none>"`.

Метод delete

```
def delete(self, key):
```

```
    if key in self.data:
        node_to_delete = self.data[key]
        if node_to_delete.prev:
            node_to_delete.prev.next = node_to_delete.next
        if node_to_delete.next:
            node_to_delete.next.prev = node_to_delete.prev
        if node_to_delete == self.head:
            self.head = node_to_delete.next
        if node_to_delete == self.tail:
            self.tail = node_to_delete.prev

        del self.data[key]
```

- Удаляет узел:
  - Обновляет указатели prev и next соседних узлов.
  - Если узел - голова или хвост, обновляются указатели head или tail.
- Удаляет ключ из словаря data.

Метод prev

```
def prev(self, key):
    if key in self.data:
        node = self.data[key].prev
        if node:
            return node.value
    return "<none>"
```

- Находит узел с ключом.
- Возвращает значение предыдущего узла или "<none>", если его нет.

Метод next

```
def next(self, key):
    if key in self.data:
        node = self.data[key].next
        if node:
            return node.value
    return "<none>"
```

- Находит узел с ключом.
- Возвращает значение следующего узла или "<none>", если его нет.

3. Функции для работы с файлами

Чтение входных данных

```
def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        operations = [f.readline().strip() for _ in range(n)]
    return operations
```

- Читает файл.
- Возвращает список операций.

Запись результатов

```
def write_output(file_path, results):
    with open(file_path, 'w') as f:
        f.write('\n'.join(results) + '\n')
```

- Записывает результаты операций в файл.

4. Функция `main`

```
def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    operations = read_input(input_path)
    associative_array = AssociativeArray()
    results = []

    for operation in operations:
        parts = operation.split()
        cmd = parts[0]
        if cmd == 'put':
            associative_array.put(parts[1], parts[2])
        elif cmd == 'get':
            results.append(associative_array.get(parts[1]))
        elif cmd == 'prev':
            results.append(associative_array.prev(parts[1]))
        elif cmd == 'next':
            results.append(associative_array.next(parts[1]))
        elif cmd == 'delete':
            associative_array.delete(parts[1])

    write_output(output_path, results)
```

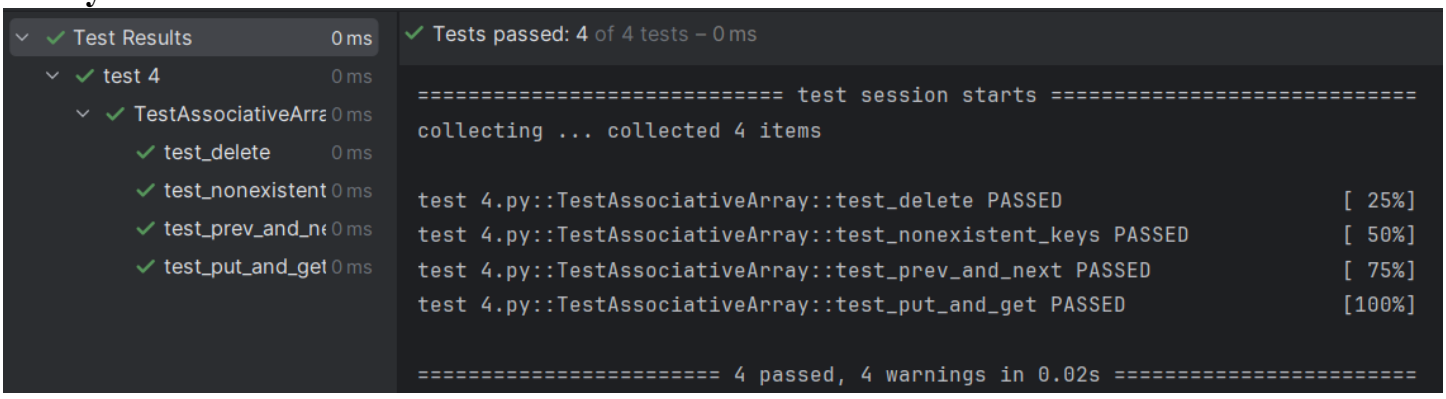
- Выполняет операции над ассоциативным массивом.
- Обрабатывает команды:
  - `put`: Добавляет или обновляет значение ключа.
  - `get`: Получает значение ключа.
  - `prev`: Находит предыдущее значение.
  - `next`: Находит следующее значение.
  - `delete`: Удаляет ключ.
- Результаты команд `get`, `prev`, `next` сохраняются в список `results`.

Unittest для задание 4:

```
import os
import sys
import unittest

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from d4 import AssociativeArray
```

```python
class TestAssociativeArray(unittest.TestCase):

    def setUp(self):
        # given
        self.associative_array = AssociativeArray()

    def test_put_and_get(self):
        # given
        self.associative_array.put("key1", "value1")
        self.assertEqual(self.associative_array.get("key1"), "value1")
        # then
        self.associative_array.put("key1", "value2")
        self.assertEqual(self.associative_array.get("key1"), "value2")

    def test_prev_and_next(self):
        # given
        self.associative_array.put("key1", "value1")
        # when
        self.associative_array.put("key2", "value2")
        self.associative_array.put("key3", "value3")
        # then
        self.assertEqual(self.associative_array.prev("key2"), "value1")
        self.assertEqual(self.associative_array.next("key2"), "value3")

    def test_delete(self):
        # given
        self.associative_array.put("key1", "value1")
        # when
        self.associative_array.delete("key1")
        # then
        self.assertEqual(self.associative_array.get("key1"), "<none>")

    def test_nonexistent_keys(self):
        # given
        self.assertEqual(self.associative_array.get("nonexistent"), "<none>")
        # when
        self.assertEqual(self.associative_array.prev("nonexistent"), "<none>")
        # then
        self.assertEqual(self.associative_array.next("nonexistent"), "<none>")

if __name__ == "__main__":
    unittest.main()
```

*** Результат :**



1. Класс `TestAssociativeArray`

Класс наследует `unittest.TestCase` и содержит тесты для проверки поведения методов.

Метод setUp

```
def setUp(self):
    self.associative_array = AssociativeArray()
```

- Выполняется перед каждым тестом.
- Создаёт новый экземпляр `AssociativeArray`.

2. Тесты

Тест test_put_and_get

```
def test_put_and_get(self):
    self.associative_array.put("key1", "value1")
    self.assertEqual(self.associative_array.get("key1"),
"value1")

    self.associative_array.put("key1", "value2")
    self.assertEqual(self.associative_array.get("key1"),
"value2")
```

- **Цель**:
  - Проверить, что метод put добавляет пару ключ-значение.
  - Убедиться, что значение обновляется, если ключ уже существует.
- **Логика**:
  - Сначала добавляется ключ key1 со значением value1.
  - Затем обновляется значение ключа key1 на value2.
- **Ожидаемый результат**:
  - get("key1") сначала возвращает value1, а затем value2.

Тест test_prev_and_next

```
def test_prev_and_next(self):
    self.associative_array.put("key1", "value1")
    self.associative_array.put("key2", "value2")
    self.associative_array.put("key3", "value3")

    self.assertEqual(self.associative_array.prev("key2"),
"value1")
    self.assertEqual(self.associative_array.next("key2"),
"value3")
```

- **Цель**:
  - Проверить, что методы prev и next возвращают корректные значения для соседних ключей.
- **Логика**:
  - Последовательно добавляются ключи key1, key2, key3.
  - Проверяется:
    - prev("key2") возвращает значение value1 (предыдущее).
    - next("key2") возвращает значение value3 (следующее).
- **Ожидаемый результат**:
  - Значения соседних ключей правильно определяются.

Тест test_delete

```
def test_delete(self):
    self.associative_array.put("key1", "value1")
    self.associative_array.delete("key1")
    self.assertEqual(self.associative_array.get("key1"),
"<none>")
```

- **Цель**:
  - Проверить корректность работы метода delete.
- **Логика**:
  - Ключ key1 добавляется, затем удаляется.
  - Проверяется, что после удаления метод get возвращает <none>.
- **Ожидаемый результат**:
  - Удалённый ключ больше не существует.

Тест test_nonexistent_keys

```
def test_nonexistent_keys(self):
    self.assertEqual(self.associative_array.get("nonexistent"),
"<none>")
    self.assertEqual(self.associative_array.prev("nonexistent"),
"<none>")
    self.assertEqual(self.associative_array.next("nonexistent"),
"<none>")
```

- **Цель**:
  - Убедиться, что методы корректно обрабатывают несуществующие ключи.
- **Логика**:
  - Проверяются методы get, prev, next для ключа, который отсутствует.
- **Ожидаемый результат**:
  - Все методы возвращают `<none>`.

3. Запуск тестов

```
if __name__ == "__main__":
    unittest.main()
```

- Запускает все тесты в классе `TestAssociativeArray`.

Задание 5 : Выборы в США

Как известно, в США президент выбирается не прямым голосованием, а путем двухуровневого голосования. Сначала проводятся выборы в каждом штате и опре- деляется победитель выборов в данном штате. Затем проводятся государственные выборы: на этих выборах каждый штат имеет определенное число голосов — чис- ло выборщиков от этого штата. На практике, все выборщики от штата голосуют в соответствии с результатами голосования внутри штата, то есть на заключитель- ной стадии выборов в голосовании участвуют штаты, имеющие различное число голосов. Вам известно за кого проголосовал каждый штат и сколько голосов бы- ло отдано данным штатом. Подведите итоги выборов: для каждого из участника голосования определите число отданных за него голосов.

- **Формат ввода / входного файла (input.txt).** Каждая строка входного файла 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, количе- ство отданных за них голосов.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 64 мб.
- Примеры:

№	input.txt	output.txt
1	McCain 10 McCain 5 Obama 9 Obama 8 McCain 1	McCain 16 Obama 17

№	input.txt	output.txt
2	ivanov 100 ivanov 500 ivanov 300 petr 70 tourist 1 tourist 2	ivanov 900 petr 70 tourist 3

№	input.txt	output.txt
3	bur 1	bur 1

```python
import os
from collections import defaultdict

def read_input(file_path):
    with open(file_path, 'r') as f:
        votes = f.readlines()
    return votes

def write_output(file_path, results):
    with open(file_path, 'w') as f:
        for candidate, total_votes in results:
            f.write(f"{candidate} {total_votes}\n")

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    votes = read_input(input_path)
    vote_count = defaultdict(int)

    for vote in votes:
        parts = vote.split()
        candidate = parts[0]
        count = int(parts[1])
        vote_count[candidate] += count

    sorted_results = sorted(vote_count.items())

    write_output(output_path, sorted_results)

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

input.txt:

```
McCain 10
McCain 5
Obama 9
Obama 8
McCain 1
```

output.txt:

```
McCain 16
Obama 17
```

1. Импорт библиотек

```
from collections import defaultdict
import os
```

- **`defaultdict`** из модуля `collections`:
  - Используется для автоматической инициализации словаря, чтобы можно было сразу складывать значения.
- **`os`**:
  - Для работы с путями к файлам.

2. Функция read_input

```
def read_input(file_path):
    with open(file_path, 'r') as f:
        votes = f.readlines()
    return votes
```

- **Цель**:
  - Считывает строки из входного файла.
- **Логика**:
  - Открывает файл, читает все строки в список и возвращает его.
- **Формат данных**:
  - Возвращает список строк, каждая строка представляет данные одного штата.

3. Функция write_output

```
def write_output(file_path, results):
    with open(file_path, 'w') as f:
        for candidate, total_votes in results:
            f.write(f"{candidate} {total_votes}\n")
```

- **Цель**:
  - Записывает результаты в выходной файл.
- **Логика**:
  - Принимает список кортежей results (кандидат, общее количество голосов).
  - Записывает каждый кортеж в файл в формате `кандидат количество`.

4. Основная функция main

```
def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')
```

```
    votes = read_input(input_path)
    vote_count = defaultdict(int)

    for vote in votes:
        parts = vote.split()
        candidate = parts[0]
        count = int(parts[1])
        vote_count[candidate] += count

    sorted_results = sorted(vote_count.items())

    write_output(output_path, sorted_results)
```

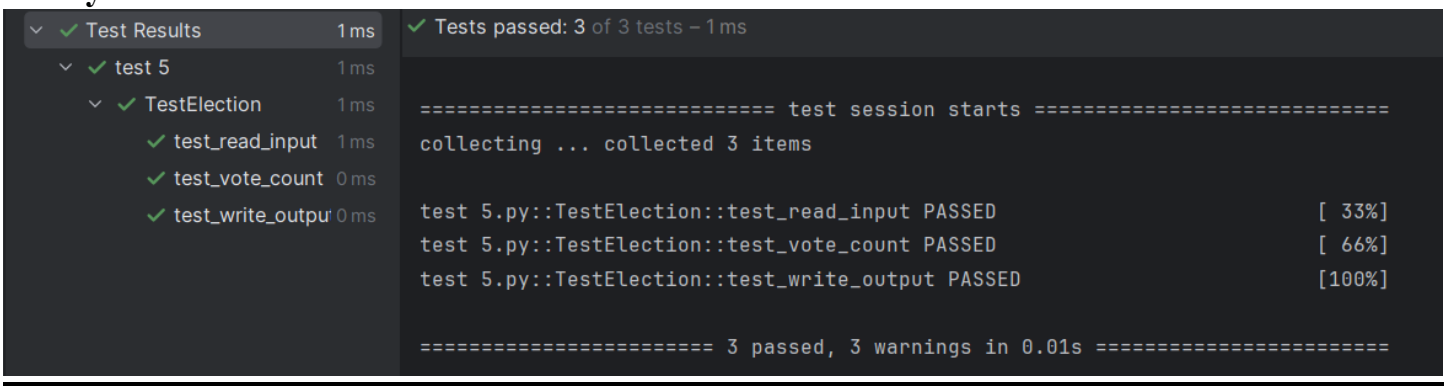
1. **Считывание данных**:
   - **read_input(input_path)**: считывает входные строки.
   - Пример строки: "bur 1".
2. **Подсчёт голосов**:
   - Используется defaultdict(int), где ключ — фамилия кандидата, а значение — сумма голосов.
   - Для каждой строки:
     - Разделяется на части с помощью split():
       - candidate: фамилия кандидата.
       - count: количество голосов выборщиков.
     - Суммируются голоса: vote_count[candidate] += count.
3. **Сортировка результатов**:
   - Используется sorted() для сортировки кандидатов в лексикографическом порядке.
   - Результат: список кортежей [(кандидат, количество)].
4. **Запись результатов**:
   - **write_output(output_path, sorted_results)** записывает данные в выходной файл.

Unittest для задание 5:

```
import os
import sys
import unittest
from collections import defaultdict

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from d5 import read_input, write_output

class TestElection(unittest.TestCase):

    def test_read_input(self):
        # given
        test_input = "McCain 10\nObama 5\nMcCain 1\n"
        with open('test_input.txt', 'w') as f:
            f.write(test_input)
```

```
        # when
        result = read_input('test_input.txt')
        expected = ['McCain 10\n', 'Obama 5\n', 'McCain 1\n']
        self.assertEqual(result, expected)

        # then
        os.remove('test_input.txt')

    def test_write_output(self):
        # given
        results = [('McCain', 16), ('Obama', 17)]
        write_output('test_output.txt', results)

        # when
        with open('test_output.txt', 'r') as f:
            content = f.readlines()

        # when
        expected_output = ["McCain 16\n", "Obama 17\n"]
        self.assertEqual(content, expected_output)

        # then
        os.remove('test_output.txt')

    def test_vote_count(self):
        # given
        votes = [
            "McCain 10\n",
            "Obama 5\n",
            "McCain 1\n"
        ]
        vote_count = defaultdict(int)

        # when
        for vote in votes:
            parts = vote.split()
            candidate = parts[0]
            count = int(parts[1])
            vote_count[candidate] += count

        # then
        expected_count = {'McCain': 11, 'Obama': 5}
        self.assertEqual(dict(vote_count), expected_count)

if __name__ == "__main__":
    unittest.main()
```

*** Результат :**



1. Импорт необходимых модулей

```
import os
import sys
import unittest
from collections import defaultdict
```

- **os**: позволяет взаимодействовать с операционной системой, например, для работы с файлами.
- **sys**: предоставляет доступ к некоторым переменным, используемым или поддерживаемым интерпретатором Python.
- **unittest**: модуль для создания и выполнения тестов.
- **defaultdict**: специальный словарь из модуля collections, который позволяет избежать ошибок при обращении к отсутствующим ключам.

2. Добавление пути к модулю

```
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from d5 import read_input, write_output
```

- Этот код добавляет путь к папке src в список путей поиска модулей, чтобы можно было импортировать функции read_input и write_output из файла d5.py.

3. Определение класса тестов

```
class TestElection(unittest.TestCase):
```

- Здесь создается класс TestElection, который наследует от unittest.TestCase. Это позволяет использовать функции для тестирования.

4. Тестирование функции `read_input`

```
def test_read_input(self):
    # given
    test_input = "McCain 10\nObama 5\nMcCain 1\n"
    with open('test_input.txt', 'w') as f:
        f.write(test_input)

    # when
    result = read_input('test_input.txt')
    expected = ['McCain 10\n', 'Obama 5\n', 'McCain 1\n']
    self.assertEqual(result, expected)

    # then
    os.remove('test_input.txt')
```

- **given**: Подготавливаем тестовые данные, создавая файл test_input.txt с данными о голосах.
- **when**: Вызываем функцию read_input, чтобы считать данные из файла.
- **then**: Сравниваем результат с ожидаемым значением и удаляем файл.

5. Тестирование функции `write_output`

```
def test_write_output(self):
    # given
    results = [('McCain', 16), ('Obama', 17)]
    write_output('test_output.txt', results)

    # when
```

```
    with open('test_output.txt', 'r') as f:
        content = f.readlines()

    # when
    expected_output = ["McCain 16\n", "Obama 17\n"]
    self.assertEqual(content, expected_output)

    # then
    os.remove('test_output.txt')
```

- **given**: Подготавливаем результаты голосования и записываем их в файл test_output.txt.
- **when**: Читаем содержимое файла.
- **then**: Сравниваем с ожидаемым выводом и удаляем файл.

6. Тестирование подсчета голосов

```
def test_vote_count(self):
    # given
    votes = [
        "McCain 10\n",
        "Obama 5\n",
        "McCain 1\n"
    ]
    vote_count = defaultdict(int)

    # when
    for vote in votes:
        parts = vote.split()
        candidate = parts[0]
        count = int(parts[1])
        vote_count[candidate] += count

    # then
    expected_count = {'McCain': 11, 'Obama': 5}
    self.assertEqual(dict(vote_count), expected_count)
```

- **given**: Создаем список голосов и инициализируем словарь для подсчета голосов.
- **when**: Считаем голоса для каждого кандидата.
- **then**: Сравниваем с ожидаемым результатом.

7. Запуск тестов

```
if __name__ == "__main__":
    unittest.main()
```

- Этот блок запускает все тесты, если файл выполняется как основная программа.

Задание 6 : Фибоначчи возвращается

Вам дается последовательность чисел. Для каждого числа определите, является ли оно числом Фибоначчи. Напомним, что числа Фибоначчи определяются, например, так:

$F_0 = F_1 = 1$ (1)

$F_i = F_{i-1} + F_{i-2}$ для $i \geq 2$.

- **Формат ввода / входного файла (input.txt).** Первая строка содержит од- но число N ($1 \le N \le 10^6$) - количество запросов. Следующие N строк содержат по одному целому числу. При этом соблюдаются следующие огра- ничения при проверке:

1. Размер каждого числа не превосходит 5000 цифр в десятичном пред- ставлении.
2. Размер входа не превышает 1 Мб.

- **Формат вывода / выходного файла (output.txt).** Для каждого числа, дан- ного во входном файле, выведите «Yes», если оно является числом Фибо- наччи, и «No» в противном случае.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 128 мб. ***Внимание:*** *есть вероятность превышения по памяти, т.к. сами по себе числа Фибоначчи большие. Делайте проверку на память!*
- Примеры:

input.txt	output.txt
8 1 2 3 4 5 6 7 8	Yes Yes Yes No Yes No No Yes

```python
import os

def is_fibonacci(num):
    if num < 0:
        return False
    a, b = 0, 1
    while b < num:
        a, b = b, a + b
    return b == num

def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        queries = [int(f.readline().strip()) for _ in range(n)]
    return queries

def write_output(file_path, results):
    with open(file_path, 'w') as f:
        for result in results:
            f.write(f"{result}\n")

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    queries = read_input(input_path)
    results = []

    for query in queries:
        if is_fibonacci(query):
```

```
            results.append("Yes")
        else:
            results.append("No")

    write_output(output_path, results)

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

input.txt :

```
8
1
2
3
4
5
6
7
8
```

output.txt:

```
Yes
Yes
Yes
No
Yes
No
No
Yes
```

1. Импорт необходимых модулей

```
import os
```

- Импортируется модуль os, который позволяет работать с файловой системой, например, для создания путей к файлам.

2. Функция `is_fibonacci`

```
def is_fibonacci(num):
    if num < 0:
        return False
    a, b = 0, 1
    while b < num:
        a, b = b, a + b
    return b == num
```

- Эта функция проверяет, является ли число Фибоначчи.
- **Проверка на отрицательные числа**: Если num меньше 0, то оно не может быть числом Фибоначчи.
- **Инициализация**: a и b инициализируются как 0 и 1 (первые два числа Фибоначчи).

- **Цикл**: Пока b меньше num, обновляем a и b на следующие числа Фибоначчи.
- **Возврат результата**: Если b равно num, значит, это число Фибоначчи, и функция возвращает True. В противном случае возвращает False.

3. Функция `read_input`

```
def read_input(file_path):
    with open(file_path, 'r') as f:
        n = int(f.readline().strip())
        queries = [int(f.readline().strip()) for _ in range(n)]
    return queries
```

- Эта функция читает входные данные из файла.
- Открывается файл по заданному пути file_path.
- Считывается первое число n, которое указывает количество запросов.
- Затем считываются n целых чисел и сохраняются в список queries.
- Функция возвращает список запросов.

4. Функция `write_output`

```
def write_output(file_path, results):
    with open(file_path, 'w') as f:
        for result in results:
            f.write(f"{result}\n")
```

- Эта функция записывает результаты в выходной файл.
- Открывается файл по указанному пути file_path для записи.
- Для каждого результата из списка results записывается строка в файл.

5. Главная функция `main`

```
def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    queries = read_input(input_path)
    results = []

    for query in queries:
        if is_fibonacci(query):
            results.append("Yes")
        else:
            results.append("No")

    write_output(output_path, results)
```

- **Определение путей**: Здесь задаются пути к входному и выходному файлам.
- **Чтение входных данных**: Вызывается функция read_input для получения списка запросов.
- **Проверка на Фибоначчи**: Для каждого запроса вызывается is_fibonacci. Результаты ("Yes" или "No") добавляются в список results.
- **Запись выходных данных**: Результаты записываются в выходной файл с помощью write_output.

6. Запуск программы

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

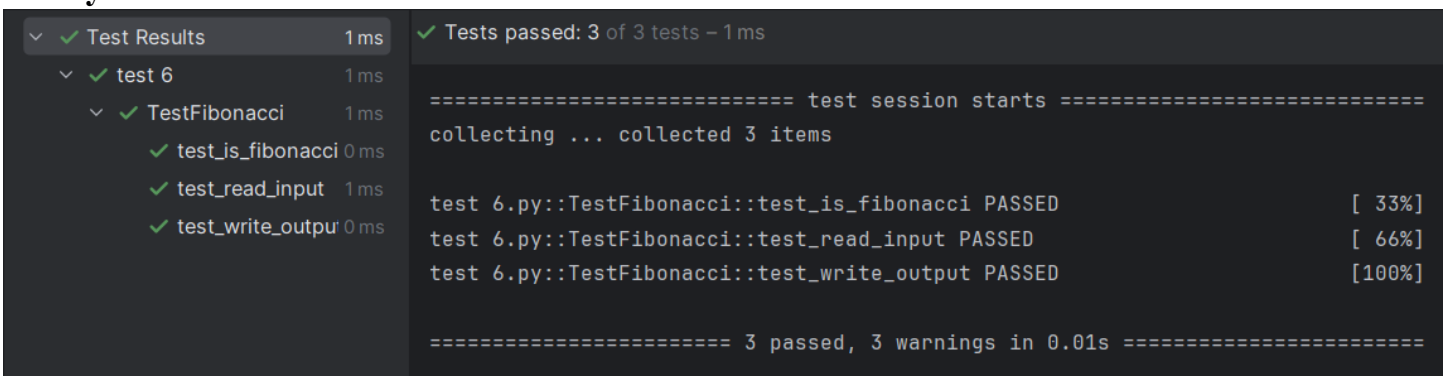
- Этот блок запускает главную функцию main, если файл выполняется как основная программа.

Unittest для задание 6:

```
import os
import sys
import unittest

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from d6 import is_fibonacci, read_input, write_output

class TestFibonacci(unittest.TestCase):

    def test_is_fibonacci(self):
        # given
        self.assertTrue(is_fibonacci(1))
        self.assertTrue(is_fibonacci(2))
        self.assertTrue(is_fibonacci(3))
        self.assertTrue(is_fibonacci(5))
        self.assertTrue(is_fibonacci(8))

        # when
        self.assertFalse(is_fibonacci(4))
        self.assertFalse(is_fibonacci(6))
        self.assertFalse(is_fibonacci(7))
        self.assertFalse(is_fibonacci(9))

        # then
        self.assertFalse(is_fibonacci(-1))

    def test_read_input(self):
        # given
        test_input = "3\n1\n2\n3\n"
        with open('test_input.txt', 'w') as f:
            f.write(test_input)

        # when
        result = read_input('test_input.txt')
        expected = [1, 2, 3]
        self.assertEqual(result, expected)

        # then
        os.remove('test_input.txt')

    def test_write_output(self):
        # given
        results = ["Yes", "Yes", "Yes"]
        write_output('test_output.txt', results)

        # when
        with open('test_output.txt', 'r') as f:
            content = f.readlines()

        # when
        expected_output = ["Yes\n", "Yes\n", "Yes\n"]
        self.assertEqual(content, expected_output)
```

```
        # then
        os.remove('test_output.txt')

if __name__ == "__main__":
    unittest.main()
```

*** Результат :**



---

1. Импорт необходимых модулей

```
import os
import sys
import unittest
```

- **os**: Модуль для работы с файловой системой.
- **sys**: Модуль, который позволяет взаимодействовать с интерпретатором Python.
- **unittest**: Модуль для создания и выполнения тестов.

2. Настройка пути к модулю

```
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src')))
from d6 import is_fibonacci, read_input, write_output
```

- Добавляется путь к директории src, чтобы можно было импортировать функции is_fibonacci, read_input и write_output из файла d6.py.

3. Определение класса тестов

```
class TestFibonacci(unittest.TestCase):
```

- Создается класс TestFibonacci, который наследует от unittest.TestCase. Это позволяет использовать методы для тестирования.

4. Тестирование функции `is_fibonacci`

```
def test_is_fibonacci(self):
    # given
    self.assertTrue(is_fibonacci(1))
    self.assertTrue(is_fibonacci(2))
    self.assertTrue(is_fibonacci(3))
    self.assertTrue(is_fibonacci(5))
    self.assertTrue(is_fibonacci(8))

    # when
    self.assertFalse(is_fibonacci(4))
    self.assertFalse(is_fibonacci(6))
```

```
    self.assertFalse(is_fibonacci(7))
    self.assertFalse(is_fibonacci(9))

    # then
    self.assertFalse(is_fibonacci(-1))
```

- **given**: Проверяем, что числа 1, 2, 3, 5 и 8 являются числами Фибоначчи с помощью assertTrue.
- **when**: Проверяем, что числа 4, 6, 7 и 9 не являются числами Фибоначчи с помощью assertFalse.
- **then**: Проверяем, что отрицательное число -1 не является числом Фибоначчи.

5. Тестирование функции `read_input`

```
def test_read_input(self):
    # given
    test_input = "3\n1\n2\n3\n"
    with open('test_input.txt', 'w') as f:
        f.write(test_input)

    # when
    result = read_input('test_input.txt')
    expected = [1, 2, 3]
    self.assertEqual(result, expected)

    # then
    os.remove('test_input.txt')
```

- **given**: Создаем тестовый файл test_input.txt с входными данными.
- **when**: Вызываем функцию read_input, чтобы считать данные из файла, и проверяем, соответствует ли результат ожидаемому списку [1, 2, 3].
- **then**: Удаляем тестовый файл после проверки.

6. Тестирование функции `write_output`

```
def test_write_output(self):
    # given
    results = ["Yes", "Yes", "Yes"]
    write_output('test_output.txt', results)

    # when
    with open('test_output.txt', 'r') as f:
        content = f.readlines()

    # when
    expected_output = ["Yes\n", "Yes\n", "Yes\n"]
    self.assertEqual(content, expected_output)

    # then
    os.remove('test_output.txt')
```

- **given**: Задаем список результатов, который будем записывать в файл test_output.txt.

- **when**: Записываем результаты в файл и затем читаем его содержимое.
- **then**: Проверяем, совпадает ли считанное содержимое с ожидаемым выводом, а затем удаляем файл.

7. Запуск тестов

```
if __name__ == "__main__":
    unittest.main()
```

- Этот блок запускает все тесты, если файл выполняется как основная программа.

Задание 7 : Драгоценные камни

В одной далекой восточной стране до сих пор по пустыням ходят караваны верблюдов, с помощью которых купцы перевозят пряности, драгоценности и до- рогие 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 со- бой. 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 обозна- чать виды камней с помощью строчных букв латинского алфавита. Шах обычно оценивает камни следующим образом. Он заранее определил несколько упоря- доченных пар типов камней: (a_1, b_1), (a_2, b_2), ..., (a_k, b_k). 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 $1 \le i < j \le n$, а камни S_i и S_j образуют красивую пару, то есть существует такое число $1 \le q \le k$, что $S_i = a_q$ и $S_j = b_q$.

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.

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла содержит целые числа n и k ($1 \le n \le 100000$, $1 \le k \le 676$) – число камней, которые привезли купцы и число пар, которые шах считает красивыми. Вто- рая 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 выве- дите ответ на задачу – количество пар, которое должен найти визирь.

- Ограничение по времени. 1 сек.
- Ограничение по памяти. 64 мб.

- Примеры:

№	input.txt	output.txt
1	7 1 abacaba aa	6

2	7 3 abacaba ab ac bb	7

```python
import os
from collections import defaultdict

def read_input(file_path):
    with open(file_path, 'r') as f:
        n, k = map(int, f.readline().strip().split())
        stones = f.readline().strip()
        pairs = [tuple(f.readline().strip()) for _ in range(k)]
    return n, k, stones, pairs

def write_output(file_path, result):
    with open(file_path, 'w') as f:
        f.write(str(result) + "\n")

def count_beautiful_pairs(n, stones, pairs):
    count = 0

    stone_count = defaultdict(int)

    for i in range(n):
        for a, b in pairs:
            if stones[i] == b:
                count += stone_count[a]
        stone_count[stones[i]] += 1

    return count

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    n, k, stones, pairs = read_input(input_path)
    result = count_beautiful_pairs(n, stones, pairs)
    write_output(output_path, result)

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

input.txt:

```
7 1
abacaba
aa
```

output.txt:

```
6
```

1. Импорт необходимых модулей

```python
import os
from collections import defaultdict
```

- **os**: Модуль для работы с файловой системой.
- **defaultdict**: Словарь с умолчанием, который позволяет удобно управлять счетчиками.

2. Функция чтения входных данных

```
def read_input(file_path):
    with open(file_path, 'r') as f:
        n, k = map(int, f.readline().strip().split())
        stones = f.readline().strip()
        pairs = [tuple(f.readline().strip()) for _ in range(k)]
    return n, k, stones, pairs
```

- Эта функция считывает данные из файла.
- Первая строка файла содержит два целых числа: n (количество камней) и k (количество красивых пар).
- Вторая строка содержит строку stones, представляющую типы камней.
- Далее считываются k строк, каждая из которых представляет красивую пару (две буквы).
- Функция возвращает n, k, строку stones и список pairs.

3. Функция записи выходных данных

```
def write_output(file_path, result):
    with open(file_path, 'w') as f:
        f.write(str(result) + "\n")
```

- Эта функция записывает результат в выходной файл. Она принимает путь к файлу и результат (число пар), который нужно записать.

4. Основная логика подсчета "красивых" пар

```
def count_beautiful_pairs(n, stones, pairs):
    count = 0
    stone_count = defaultdict(int)

    for i in range(n):
        for a, b in pairs:
            if stones[i] == b:
                count += stone_count[a]
        stone_count[stones[i]] += 1

    return count
```

- **Инициализация**: count используется для подсчета красивых пар, а stone_count — для хранения количества камней, встреченных до текущего индекса.
- **Цикл по камням**: Проходим по каждому камню в строке stones:
  - Для каждой красивой пары (a, b) проверяем, является ли текущий камень stones[i] равным b.
  - Если да, добавляем к count количество камней типа a, которые были встречены ранее (это делается с помощью stone_count[a]).
  - Увеличиваем счетчик для текущего камня stones[i] в stone_count.
- Функция возвращает общее количество красивых пар.

5. Главная функция

```
def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')
```

```
    n, k, stones, pairs = read_input(input_path)
    result = count_beautiful_pairs(n, stones, pairs)
    write_output(output_path, result)
```

- В этой функции задаются пути к входному и выходному файлам.
- Вызывается функция read_input для чтения данных.
- Затем выполняется подсчет красивых пар с помощью count_beautiful_pairs.
- Наконец, результат записывается в выходной файл с помощью write_output.

6. Запуск программы

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

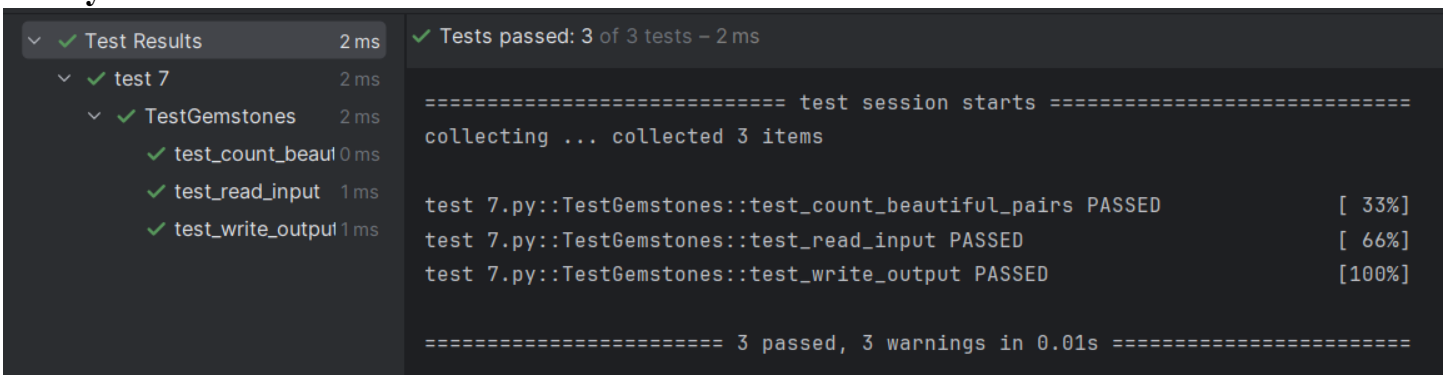
- Этот блок запускает главную функцию main, если файл выполняется как основная программа.

Unittest для задание 7:

```
import os
import sys
import unittest

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from d7 import read_input, write_output, count_beautiful_pairs

class TestGemstones(unittest.TestCase):

    def test_count_beautiful_pairs(self):
        # given
        stones = "abacaba"
        pairs = [("a", "a")]
        result = count_beautiful_pairs(len(stones), stones, pairs)
        self.assertEqual(result, 6)

        # then
        pairs = [("a", "b"), ("a", "c"), ("b", "b")]
        result = count_beautiful_pairs(len(stones), stones, pairs)
        self.assertEqual(result, 7)

    def test_read_input(self):
        # given
        test_input = "7 1\nabacaba\naa\n"
        with open('test_input.txt', 'w') as f:
            f.write(test_input)

        # when
        n, k, stones, pairs = read_input('test_input.txt')
        self.assertEqual(n, 7)
        self.assertEqual(k, 1)
        self.assertEqual(stones, "abacaba")
        self.assertEqual(pairs, [("a", "a")])

        # then
        os.remove('test_input.txt')

    def test_write_output(self):
        # given
        write_output('test_output.txt', 6)
```

```
        # when
        with open('test_output.txt', 'r') as f:
            content = f.read().strip()

        # when
        self.assertEqual(content, "6")

        # then
        os.remove('test_output.txt')

if __name__ == "__main__":
    unittest.main()
```

*** Результат :**



1. Импорт необходимых модулей

```
import os
import sys
import unittest
```

- **os**: Модуль для работы с файловой системой.
- **sys**: Модуль для взаимодействия с интерпретатором Python.
- **unittest**: Модуль для создания и выполнения тестов.

2. Настройка пути к модулю

```
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src')))
from d7 import read_input, write_output, count_beautiful_pairs
```

- Добавляем путь к директории src, чтобы можно было импортировать функции read_input, write_output и count_beautiful_pairs из файла d7.py.

3. Определение класса тестов

```
class TestGemstones(unittest.TestCase):
```

- Создается класс TestGemstones, который наследует от unittest.TestCase. Это позволяет использовать методы для тестирования.

4. Тестирование функции `count_beautiful_pairs`

```
def test_count_beautiful_pairs(self):
    # given
    stones = "abacaba"
    pairs = [("a", "a")]
    result = count_beautiful_pairs(len(stones), stones, pairs)
    self.assertEqual(result, 6)
```

```
    # then
    pairs = [("a", "b"), ("a", "c"), ("b", "b")]
    result = count_beautiful_pairs(len(stones), stones, pairs)
    self.assertEqual(result, 7)
```

- **given**: Подготавливаем строку stones и список пар pairs. Тестируем, что для пары ("a", "a") результат равен 6.
- **then**: Меняем пары на ("a", "b"), ("a", "c"), ("b", "b") и проверяем, что результат равен 7.

5. Тестирование функции `read_input`

```
def test_read_input(self):
    # given
    test_input = "7 1\nabacaba\naa\n"
    with open('test_input.txt', 'w') as f:
        f.write(test_input)

    # when
    n, k, stones, pairs = read_input('test_input.txt')
    self.assertEqual(n, 7)
    self.assertEqual(k, 1)
    self.assertEqual(stones, "abacaba")
    self.assertEqual(pairs, [("a", "a")])

    # then
    os.remove('test_input.txt')
```

- **given**: Создаем тестовый файл test_input.txt с входными данными.
- **when**: Вызываем функцию read_input, чтобы считать данные из файла, и проверяем, соответствует ли результат ожидаемым значениям.
- **then**: Удаляем тестовый файл после проверки.

6. Тестирование функции `write_output`

```
def test_write_output(self):
    # given
    write_output('test_output.txt', 6)

    # when
    with open('test_output.txt', 'r') as f:
        content = f.read().strip()

    # when
    self.assertEqual(content, "6")

    # then
    os.remove('test_output.txt')
```

- **given**: Записываем результат 6 в файл test_output.txt.
- **when**: Читаем содержимое файла и проверяем, соответствует ли оно строке "6".
- **then**: Удаляем тестовый файл.

7. Запуск тестов

```
if __name__ == "__main__":
    unittest.main()
```

- Этот блок запускает все тесты, если файл выполняется как основная программа.

Задание 8 : Почти интерактивная хеш-таблица

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 $[0; 10^{15} - 1]$. 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:

- $1 \le N \le 10^7$
- $1 \le X \le 10^{15}$
- $1 \le A \le 10^3$
- $1 \le B \le 10^{15}$

Требуется N раз выполнить следующую последовательность операций:

- Если X содержится в таблице, то установить $A \leftarrow (A + A_C) \bmod 10^3$, $B \leftarrow (B + B_C) \bmod 10^{15}$.
- Если X не содержится в таблице, то добавить X в таблицу и установить $A \leftarrow (A + A_D) \bmod 10^3$, $B \leftarrow (B + B_D) \bmod 10^{15}$.
- Установить $X \leftarrow (X \cdot A + B) \bmod 10^{15}$.

Начальные значения X, A и B, а также N, A_C, B_C, A_D и B_D даны во входном файле. Выведите значения X, A и B после окончания работы.

- **Формат входного файла (input.txt).** В первой строке входного файла со- держится четыре целых числа N, X, A, B. Во второй строке содержится еще четыре целых числа A_C, B_C, A_D и B_D такие что $0 \le A_C, A_D < 10^3$, $0 \le B_C, B_D < 10^{15}$.
- **Формат выходного файла (output.txt).** Выведите значения X, A и B после окончания работы.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt
4 0 0 0 1 1 0 0
output.txt
3 1 1

```
import os

class HashTable:
    def __init__(self):
        self.table = set()
```

```
    def add(self, x):
        self.table.add(x)

    def contains(self, x):
        return x in self.table

def read_input(file_path):
    with open(file_path, 'r') as f:
        N, X, A, B = map(int, f.readline().strip().split())
        AC, BC, AD, BD = map(int, f.readline().strip().split())
    return N, X, A, B, AC, BC, AD, BD

def write_output(file_path, X, A, B):
    with open(file_path, 'w') as f:
        f.write(f"{X} {A} {B}\n")

def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    N, X, A, B, AC, BC, AD, BD = read_input(input_path)

    hash_table = HashTable()

    for _ in range(N):
        if hash_table.contains(X):
            A = (A + AC) % 1000
            B = (B + BC) % (10**15)
        else:
            hash_table.add(X)
            A = (A + AD) % 1000
            B = (B + BD) % (10**15)

        X = (X * A + B) % (10**15)

    write_output(output_path, X, A, B)

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

input.txt:

```
4 0 0 0
1 1 0 0
```

output.txt:

```
3 1 1
```

1. Определение класса `HashTable`

```
class HashTable:
    def __init__(self):
        self.table = set()

    def add(self, x):
        self.table.add(x)

    def contains(self, x):
        return x in self.table
```

- **__init__**: Конструктор инициализирует хеш-таблицу как множество (set), что позволяет эффективно хранить и проверять наличие чисел.
- **add**: Метод для добавления числа x в хеш-таблицу.
- **contains**: Метод, который проверяет, содержится ли число x в хеш-таблице.

2. Функция чтения входных данных

```
def read_input(file_path):
    with open(file_path, 'r') as f:
        N, X, A, B = map(int, f.readline().strip().split())
        AC, BC, AD, BD = map(int, f.readline().strip().split())
    return N, X, A, B, AC, BC, AD, BD
```

- Эта функция считывает данные из файла.
- Первой строкой считываются значения N, X, A, B.
- Второй строкой считываются значения AC, BC, AD, BD.
- Функция возвращает все эти значения.

3. Функция записи выходных данных

```
def write_output(file_path, X, A, B):
    with open(file_path, 'w') as f:
        f.write(f"{X} {A} {B}\n")
```

- Эта функция записывает значения X, A и B в выходной файл.

4. Главная функция

```
def main():
    input_path = os.path.join('..', 'txtf', 'input.txt')
    output_path = os.path.join('..', 'txtf', 'output.txt')

    N, X, A, B, AC, BC, AD, BD = read_input(input_path)

    hash_table = HashTable()

    for _ in range(N):
        if hash_table.contains(X):
            A = (A + AC) % 1000
            B = (B + BC) % (10**15)
        else:
            hash_table.add(X)
            A = (A + AD) % 1000
            B = (B + BD) % (10**15)

        X = (X * A + B) % (10**15)

    write_output(output_path, X, A, B)
```

- **Пути к файлам**: Задаются пути к входному и выходному файлам.
- **Чтение данных**: Вызывается функция read_input для получения значений.
- **Создание хеш-таблицы**: Инициализируется объект hash_table класса HashTable.
- **Цикл по N**: Выполняется N итераций, в каждой из которых:

- Если X содержится в хеш-таблице, обновляются значения A и B с использованием AC и BC.
- Если X не содержится, добавляется в хеш-таблицу и обновляются A и B с использованием AD и BD.
- Обновляется значение X согласно формуле.

- **Запись результата**: После выполнения всех операций результаты записываются в выходной файл с помощью функции write_output.

5. Запуск программы

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

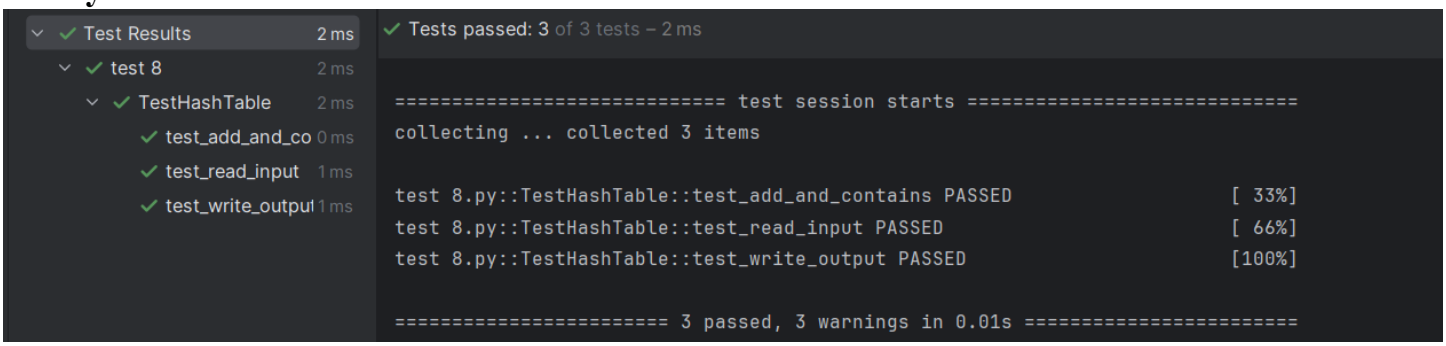
- Этот блок запускает главную функцию main, если файл выполняется как основная программа.

Unittest для задание 8:

```
import os
import sys
import unittest

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'src')))
from d8 import read_input, write_output, HashTable

class TestHashTable(unittest.TestCase):

    def test_add_and_contains(self):
        # given
        hash_table = HashTable()
        self.assertFalse(hash_table.contains(5))

        # when
        hash_table.add(5)
        self.assertTrue(hash_table.contains(5))

        # then
        hash_table.add(10)
        self.assertTrue(hash_table.contains(10))
        self.assertFalse(hash_table.contains(15))

    def test_read_input(self):
        # given
        test_input = "4 0 0 0\n1 1 0 0\n"
        with open('test_input.txt', 'w') as f:
            f.write(test_input)

        # when
        N, X, A, B, AC, BC, AD, BD = read_input('test_input.txt')
        self.assertEqual(N, 4)
        self.assertEqual(X, 0)
        self.assertEqual(A, 0)
        self.assertEqual(B, 0)
        self.assertEqual(AC, 1)
        self.assertEqual(BC, 1)
        self.assertEqual(AD, 0)
        self.assertEqual(BD, 0)

        # then
        os.remove('test_input.txt')
```

```
    def test_write_output(self):
        # given
        write_output('test_output.txt', 3, 1, 1)

        # when
        with open('test_output.txt', 'r') as f:
            content = f.read().strip()

        # when
        self.assertEqual(content, "3 1 1")

        # then
        os.remove('test_output.txt')

if __name__ == "__main__":
    unittest.main()
```

*** Результат :**



1. Импорт необходимых модулей

```
import os
import sys
import unittest
```

- **os**: Модуль для работы с файловой системой.
- **sys**: Модуль для взаимодействия с интерпретатором Python.
- **unittest**: Модуль для создания и выполнения тестов.

2. Настройка пути к модулю

```
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'src')))
from d8 import read_input, write_output, HashTable
```

- Добавляется путь к директории src, чтобы можно было импортировать функции read_input, write_output и класс HashTable из файла d8.py.

3. Определение класса тестов

```
class TestHashTable(unittest.TestCase):
```

- Создается класс TestHashTable, который наследует от unittest.TestCase. Это позволяет использовать методы для тестирования.

4. Тестирование методов `add` и `contains` хеш-таблицы

```
def test_add_and_contains(self):
    # given
    hash_table = HashTable()
    self.assertFalse(hash_table.contains(5))
```

```
    # when
    hash_table.add(5)
    self.assertTrue(hash_table.contains(5))

    # then
    hash_table.add(10)
    self.assertTrue(hash_table.contains(10))
    self.assertFalse(hash_table.contains(15))
```

- **given**: Создаем экземпляр хеш-таблицы и проверяем, что число 5 не содержится в ней (должно вернуть False).
- **when**: Добавляем число 5 в хеш-таблицу и проверяем, что теперь оно содержится (должно вернуть True).
- **then**: Добавляем число 10 и проверяем, что оно также содержится. Затем проверяем, что число 15 не содержится в хеш-таблице (должно вернуть False).

5. Тестирование функции `read_input`

```
def test_read_input(self):
    # given
    test_input = "4 0 0 0\n1 1 0 0\n"
    with open('test_input.txt', 'w') as f:
        f.write(test_input)

    # when
    N, X, A, B, AC, BC, AD, BD = read_input('test_input.txt')
    self.assertEqual(N, 4)
    self.assertEqual(X, 0)
    self.assertEqual(A, 0)
    self.assertEqual(B, 0)
    self.assertEqual(AC, 1)
    self.assertEqual(BC, 1)
    self.assertEqual(AD, 0)
    self.assertEqual(BD, 0)

    # then
    os.remove('test_input.txt')
```

- **given**: Создаем тестовый файл test_input.txt с входными данными.
- **when**: Вызываем функцию read_input, чтобы считать данные из файла, и проверяем, соответствуют ли считанные значения ожидаемым.
- **then**: Удаляем тестовый файл после проверки.

6. Тестирование функции `write_output`

```
def test_write_output(self):
    # given
    write_output('test_output.txt', 3, 1, 1)

    # when
    with open('test_output.txt', 'r') as f:
```

```
        content = f.read().strip()

    # when
    self.assertEqual(content, "3 1 1")

    # then
    os.remove('test_output.txt')
```

- **given**: Записываем значения 3, 1, 1 в файл test_output.txt.
- **when**: Читаем содержимое файла и проверяем, соответствует ли оно строке "3 1 1".
- **then**: Удаляем тестовый файл.

7. Запуск тестов

```
if __name__ == "__main__":
    unittest.main()
```

- Этот блок запускает все тесты, если файл выполняется как основная программа.