

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 5

Выполнил:
Артемов И. В.
К3141

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Множество	3
Задача №2: Телефонная книга	7
Задача №4. Прошитый ассоциативный массив	13
Задача №7. Драгоценные камни	21
Вывод	26

Задачи по варианту

Задача №1. Множество

Текст задачи.

1 задача. Множество

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

- **Формат входного файла (input.txt).** В первой строке входного файла находится строго положительное целое число операций N , не превышающее $5 \cdot 10^5$. В каждой из последующих N строк находится одна из следующих операций:
 - $A\ x$ – добавить элемент x в множество. Если элемент уже есть в множестве, 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 результат 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	Y
A 2	N
A 5	N
A 3	
? 2	
? 4	
A 2	
D 2	
? 2	

Листинг кода.

```
# Импортируем библиотеки для отслеживания памяти и времени выполнения
программы
import tracemalloc
import time
import os
from lab6.utils import open_file, write_file, delete_prev_values,
get_output_path, print_output_file

# Запускаем таймер для измерения времени работы программы
t_start = time.perf_counter()

# Включаем отслеживание памяти
tracemalloc.start()
```

```

# Устанавливаем пути к файлам
current_dir = os.path.dirname(os.path.abspath(__file__)) # Директория
текущего скрипта
txtf_dir = os.path.join(os.path.dirname(current_dir), "txtf") # Директория
с текстовыми файлами
input_path = os.path.join(txtf_dir, "input.txt")

# Функция обработки операций множества
def process_operations(n, operations):
    current_set = set() # Создаём множество для хранения элементов
    results = [] # Для хранения результатов операций "?"

    for operation in operations:
        cmd, x = operation.split()
        x = int(x)

        if cmd == "A":
            # Добавляем элемент x в множество
            current_set.add(x)
        elif cmd == "D":
            # Удаляем элемент x из множества, если он существует
            current_set.discard(x)
        elif cmd == "?":
            # Проверяем наличие элемента x и записываем результат
            if x in current_set:
                results.append("Y")
            else:
                results.append("N")
    return results

# Основной блок программы
if __name__ == "__main__":
    # Читаем данные из файла input.txt с помощью функции open_file
    data = open_file(input_path)
    n = int(data[0]) # Первая строка содержит количество операций
    operations = data[1:] # Последующие строки – это операции

    # Проверка корректности входных данных
    if 1 <= n <= 5 * 10**5 and all(op[0] in "AD?" and abs(int(op[2:])) <=
10**18 for op in operations):
        print(f"\nTask: 1\nInput:\n{n}\n\n{operations}")
        delete_prev_values(1)

        # Обрабатываем операции
        results = process_operations(n, operations)

        # Записываем результаты в файл output.txt
        output_path = get_output_path(1)
        write_file("\n".join(results), output_path)
        print_output_file(1)
    else:
        # Выводим сообщение об ошибке, если данные некорректны
        print('Введите корректные данные')

    # Выводим время работы программы
    print("Время работы: %s секунд" % (time.perf_counter() - t_start))
    # Выводим количество памяти, затраченной на выполнение программы
    print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")

    # Останавливаем отслеживание памяти
    tracemalloc.stop()

```

Текстовое объяснение решения.

1. Импортируются библиотеки для отслеживания памяти и времени.
2. Запускается таймер для измерения времени работы программы.
3. Включается отслеживание памяти.
4. Устанавливаются пути к файлам.
5. Определяется функция `process_operations`, которая выполняет операции над множеством.
6. В основном блоке программы:
 - Читаются данные из файла `input.txt`.
 - В первой строке извлекается количество операций `n`.
 - В последующих строках извлекаются операции.
7. Проверяется корректность входных данных (ограничения на количество операций и их формат).
8. Если данные корректны, выводится информация о задаче и входных данных.
9. Выполняются операции над множеством с помощью функции `process_operations`.
10. Результаты операций записываются в файл `output.txt`.
11. Выводится информация о времени работы программы.
12. Выводится информация о затраченной памяти.
13. Останавливается отслеживание памяти.

Результат работы кода на примере из задачи:

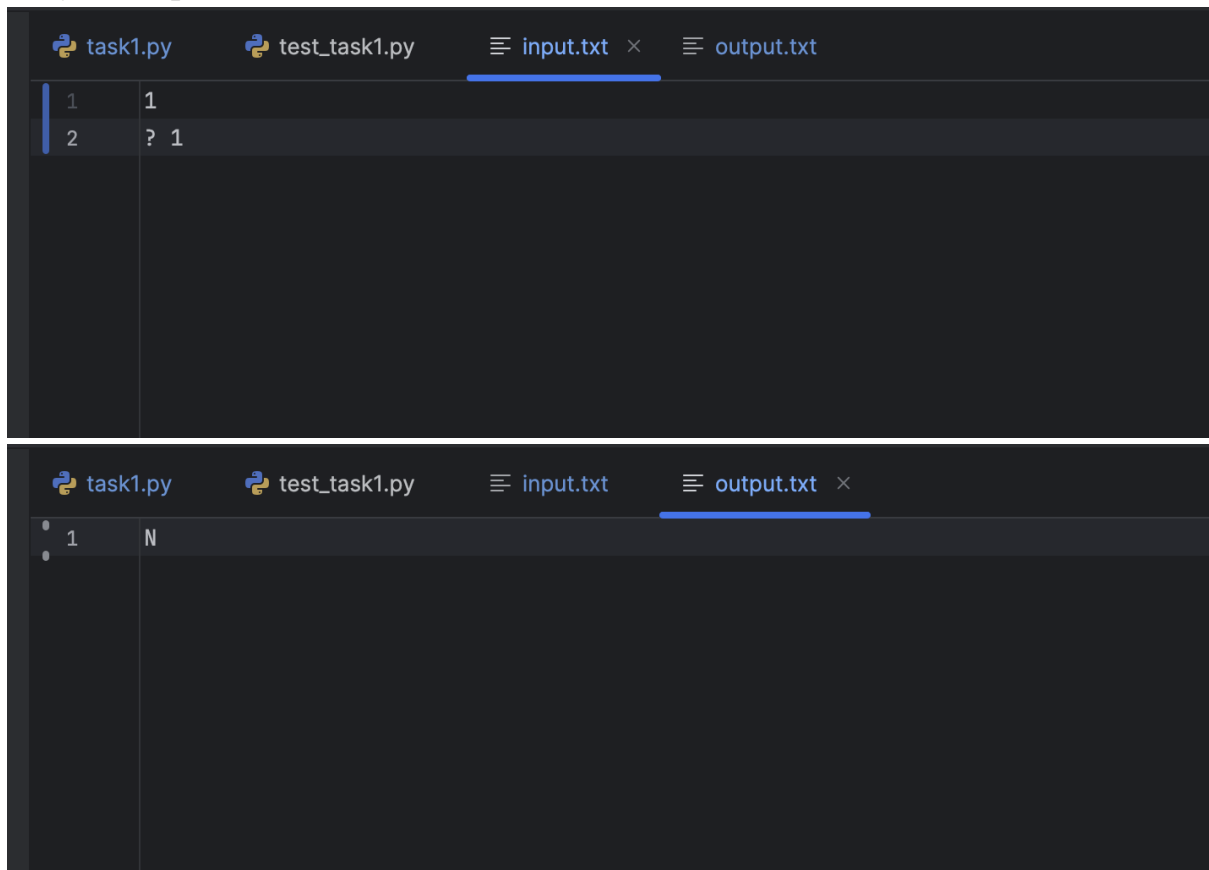
The image shows two screenshots of a code editor interface. The top screenshot displays the `input.txt` file with the following content:

```
1 8
2 A 2
3 A 5
4 A 3
5 ? 2
6 ? 4
7 A 2
8 D 2
9 ? 2
```

The bottom screenshot displays the `output.txt` file with the following content:

```
1 Y
2 N
3 N
```

Результат работы кода на максимальных и минимальных значениях:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0007107499995981925 секунд	15368 байт
Пример из задачи	0.0006332500006465125 секунд	15715 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.4109084579977207 секунд	10765530 байт

Вывод по задаче: мною был изучен алгоритм реализации множества с некоторыми операциями.

Задача №2: Телефонная книга

Текст задачи.

2 задача. Телефонная книга

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

- `add number name` – это команда означает, что пользователь добавляет в телефонную книгу человека с именем `name` и номером телефона `number`. Если пользователь с таким номером уже существует, то ваш менеджер должен перезаписать соответствующее имя.
- `del number` – означает, что менеджер должен удалить человека с номером из телефонной книги. Если такого человека нет, то он должен просто игнорировать запрос.
- `find number` – означает, что пользователь ищет человека с номером телефона `number`. Менеджер должен ответить соответствующим именем или строкой «not found» (без кавычек), если такого человека в книге нет.

- **Формат ввода / входного файла (input.txt).** В первой строке входного файла содержится число N ($1 \leq N \leq 10^5$) - 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 каждого поискового запроса `find` – имя, соответствующее номеру телефона, или «not found» (без кавычек), если в телефонной книге нет человека с таким номером телефона. Выведите по одному результату в каждой строке в том же порядке, как были заданы запросы типа `find` во входных данных.
- Ограничение по времени. 6 сек.
- Ограничение по памяти. 512 мб.
- Примеры:

1:	input.txt	
	12	
	add 911 police	
	add 76213 Mom	
	add 17239 Bob	
	find 76213	
	find 910	
	find 911	
	del 910	
	del 911	
	2:	
	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».

Листинг кода.

```
import tracemalloc
import time
import os
from lab6.utils import open_file, write_file, get_output_path,
delete_prev_values, print_output_file

# Запускаем таймер для измерения времени работы программы
t_start = time.perf_counter()

# Включаем отслеживание памяти
tracemalloc.start()

# Пути к входному и выходному файлам
current_dir = os.path.dirname(os.path.abspath(__file__)) # Текущая
директория
txtf_dir = os.path.join(os.path.dirname(current_dir), "txtf") # Директория
txtf
input_path = os.path.join(txtf_dir, "input.txt")

def is_valid_name(name):
    """
```



```

        Проверяет валидность имени.
        Имя должно содержать только латинские буквы и иметь длину не более 15
        символов.
        """
        return name.isalpha() and len(name) <= 15

def is_valid_number(number):
    """
    Проверяет валидность номера телефона.
    Номер должен состоять из цифр, иметь длину от 1 до 7 символов и не
    начинаться с нуля.
    """
    return number.isdigit() and 1 <= len(number) <= 7 and number[0] != '0'

def process_phone_book(queries):
    """
    Обрабатывает запросы для телефонной книги.

    :param queries: Список запросов
    :return: Список результатов для запросов типа "find"
    """
    phone_book = {} # Словарь для хранения телефонной книги
    results = [] # Список результатов для команд "find"

    for query in queries:
        command = query.split()
        if command[0] == "add":
            number, name = command[1], command[2]
            if is_valid_number(number) and is_valid_name(name):
                phone_book[number] = name
            else:
                results.append("invalid") # Для отладки можно использовать
                # вывод ошибки
        elif command[0] == "del":
            number = command[1]
            if is_valid_number(number):
                phone_book.pop(number, None)
        elif command[0] == "find":
            number = command[1]
            if is_valid_number(number):
                results.append(phone_book.get(number, "not found"))
            else:
                results.append("not found") # Некорректный номер считается
                # не найденным

    return results

# Основной блок программы
if __name__ == "__main__":
    # Читаем данные из файла input.txt
    lines = open_file(input_path)
    n = int(lines[0].strip()) # Количество запросов
    queries = lines[1:] # Список запросов

    # Проверка корректности входных данных
    if 1 <= n <= 10 ** 5:
        print(f"\nTask 2\nInput:\n{n}\n{queries}")
        delete_prev_values(1)

    # Обрабатываем запросы
    results = process_phone_book(queries)

```

```

# Формируем путь к выходному файлу
output_path = get_output_path(1)

# Записываем результаты в файл output.txt
write_file("\n".join(results), output_path)
print_output_file(1)

else:
    print("Введите корректные данные")

# Выводим время работы программы
print("Время работы: %s секунд" % (time.perf_counter() - t_start))
# Выводим количество памяти, затраченной на выполнение программы
print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")

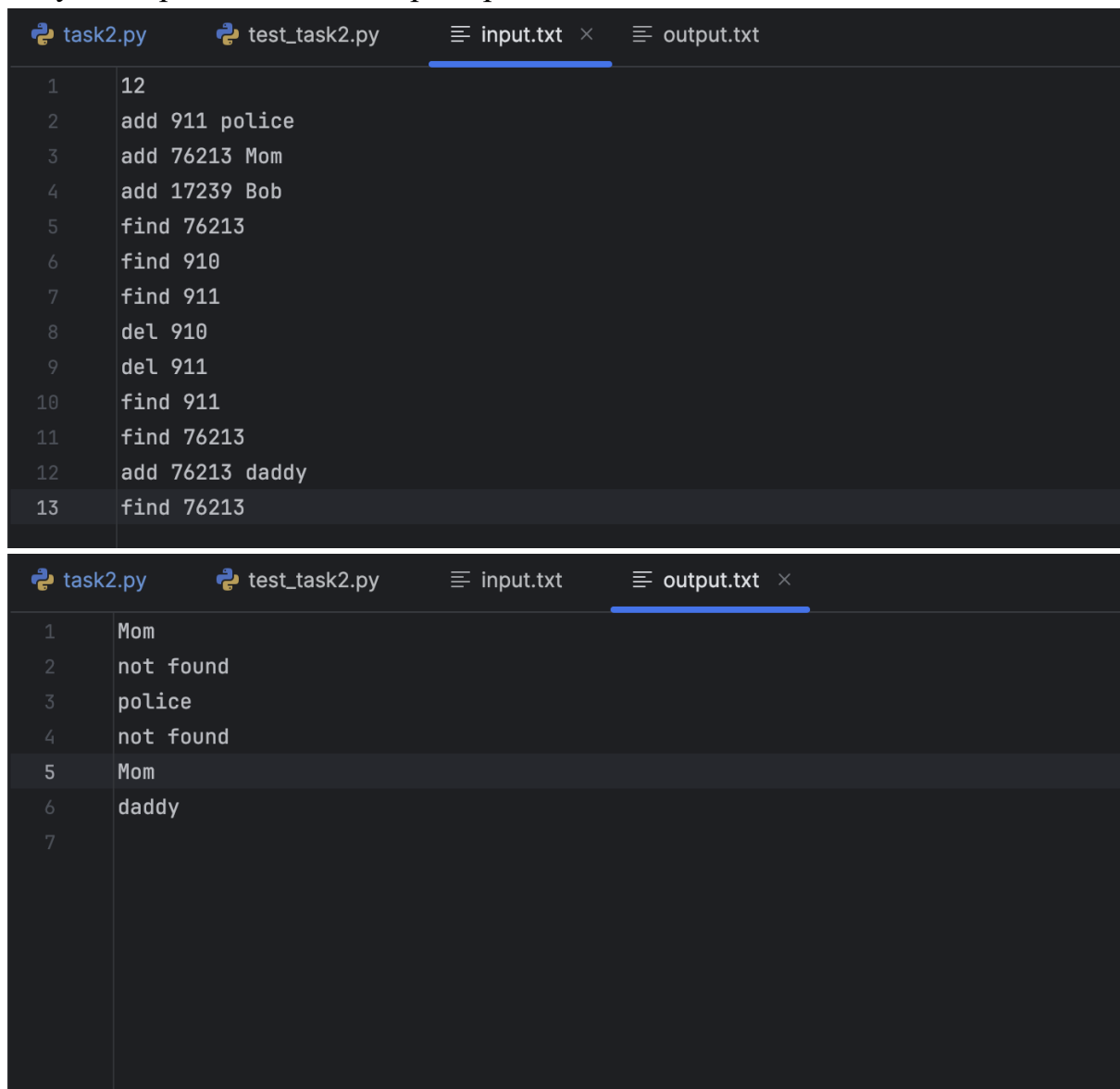
# Останавливаем отслеживание памяти
tracemalloc.stop()

```

Текстовое объяснение решения.

1. Импортируются библиотеки для отслеживания памяти и времени.
2. Запускается таймер для измерения времени работы программы.
3. Включается отслеживание памяти.
4. Устанавливаются пути к файлам.
5. Определяются функции:
 - `is_valid_name` для проверки валидности имени.
 - `is_valid_number` для проверки валидности номера телефона.
 - `process_phone_book` для обработки запросов телефонной книги.
6. В основном блоке программы:
 - Читаются данные из файла `input.txt`.
 - В первой строке извлекается количество запросов `n`.
 - В последующих строках извлекаются запросы.
7. Проверяется корректность входных данных (ограничения на количество запросов).
8. Если данные корректны, выводится информация о задаче и входных данных.
9. Обрабатываются запросы с помощью функции `process_phone_book`.
10. Формируется путь к выходному файлу.
11. Результаты записываются в файл `output.txt`.
12. Выводится информация о времени работы программы.
13. Выводится информация о затраченной памяти.
14. Останавливается отслеживание памяти.

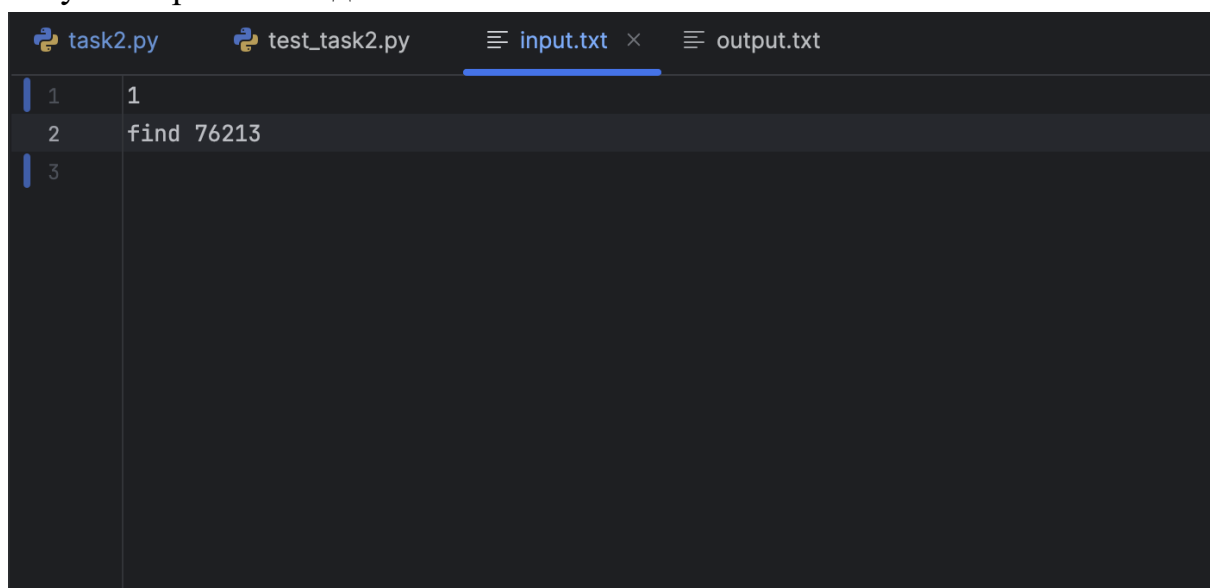
Результат работы кода на примере из текста задачи:



The screenshot shows a code editor with two tabs: `task2.py` and `test_task2.py`. The `input.txt` tab is active, displaying 13 lines of input commands. The `output.txt` tab is also visible, showing the corresponding output results.

Line	Input	Output
1	12	Mom
2	add 911 police	not found
3	add 76213 Mom	police
4	add 17239 Bob	not found
5	find 76213	Mom
6	find 910	
7	find 911	
8	del 910	
9	del 911	
10	find 911	
11	find 76213	
12	add 76213 daddy	daddy
13	find 76213	

Результат работы кода на максимальных и минимальных значениях:



The screenshot shows a code editor with two tabs: `task2.py` and `test_task2.py`. The `input.txt` tab is active, displaying 3 lines of input commands. The `output.txt` tab is also visible, showing the corresponding output results.

Line	Input	Output
1	1	
2	find 76213	
3		

```
task2.py test_task2.py input.txt output.txt x
1 not found
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0006144999997559353 секунд	15664 байт
Пример из задачи	0.000861916000758356 секунд	16335 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.4109084579977207 секунд	430401 байт

Вывод по задаче: мною был изучен алгоритм реализации простого менеджера телефонной книги.

Задача №4. Прошитый ассоциативный массив

Текст задачи.

4 задача. Прошитый ассоциативный массив

Реализуйте прошитый ассоциативный массив. Ваш алгоритм должен поддерживать 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 ассоциативном 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 ассоциативном 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 оказывается 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 находится 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 результат 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	c
put zero a	b
put one b	d
put two c	c
put three d	a
put four e	e
get two	<none>
prev two	
next two	
delete one	
delete three	
get two	
prev two	
next two	
next four	

- P.s. Задача на [openedu](#), 8 неделя.

Листинг кода.

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

# Запускаем таймер для измерения времени работы программы
t_start = time.perf_counter()

# Включаем отслеживание памяти
tracemalloc.start()

current_dir = os.path.dirname(os.path.abspath(__file__)) # Директория
task/src
txtf_dir = os.path.join(os.path.dirname(current_dir), "txtf") # Директория
task/txtf
input_path = os.path.join(txtf_dir, "input.txt")

def process_commands(commands):
    """
    Обработывает команды работы с ассоциативным массивом.
    """
    assoc_array = OrderedDict()
    results = []

    for line in commands:
        parts = line.strip().split()
        command = parts[0]

        if command == "put":
```

```

        # Добавление или обновление ключа
        x, y = parts[1], parts[2]
        if x in assoc_array:
            assoc_array[x] = y
        else:
            assoc_array[x] = y

    elif command == "get":
        # Получение значения по ключу
        x = parts[1]
        results.append(assoc_array.get(x, "<none>"))

    elif command == "prev":
        # Поиск предыдущего ключа
        x = parts[1]
        if x in assoc_array:
            keys = list(assoc_array.keys())
            idx = keys.index(x)
            if idx > 0:
                results.append(assoc_array[keys[idx - 1]])
            else:
                results.append("<none>")
        else:
            results.append("<none>")

    elif command == "next":
        # Поиск следующего ключа
        x = parts[1]
        if x in assoc_array:
            keys = list(assoc_array.keys())
            idx = keys.index(x)
            if idx < len(keys) - 1:
                results.append(assoc_array[keys[idx + 1]])
            else:
                results.append("<none>")
        else:
            results.append("<none>")

    elif command == "delete":
        # Удаление ключа
        x = parts[1]
        assoc_array.pop(x, None)

    return results

# Основной блок программы
if __name__ == "__main__":
    # Читаем данные из файла input.txt с помощью функции open_file
    lines = open_file(input_path)
    n = int(lines[0]) # Преобразуем первую строку в число операций
    commands = lines[1:] # Команды операций

    # Проверяем корректность входных данных
    if 1 <= n <= 5 * 10 ** 5 and all(len(cmd.split()) in (2, 3) for cmd in
commands):
        print(f"\nTask 4\nInput:\n{lines}")
        delete_prev_values(4)

        # Обрабатываем команды
        results = process_commands(commands)

        # Записываем результаты в файл output.txt
        output_path = get_output_path(4)

```

```

write_file("\n".join(results), output_path)
print_output_file(4)
else:
    # Выводим сообщение об ошибке, если данные некорректны
    print("Введите корректные данные")

# Выводим время работы программы
print("Время работы: %s секунд" % (time.perf_counter() - t_start))
# Выводим количество памяти, затраченной на выполнение программы
print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")

# Останавливаем отслеживание памяти
tracemalloc.stop()

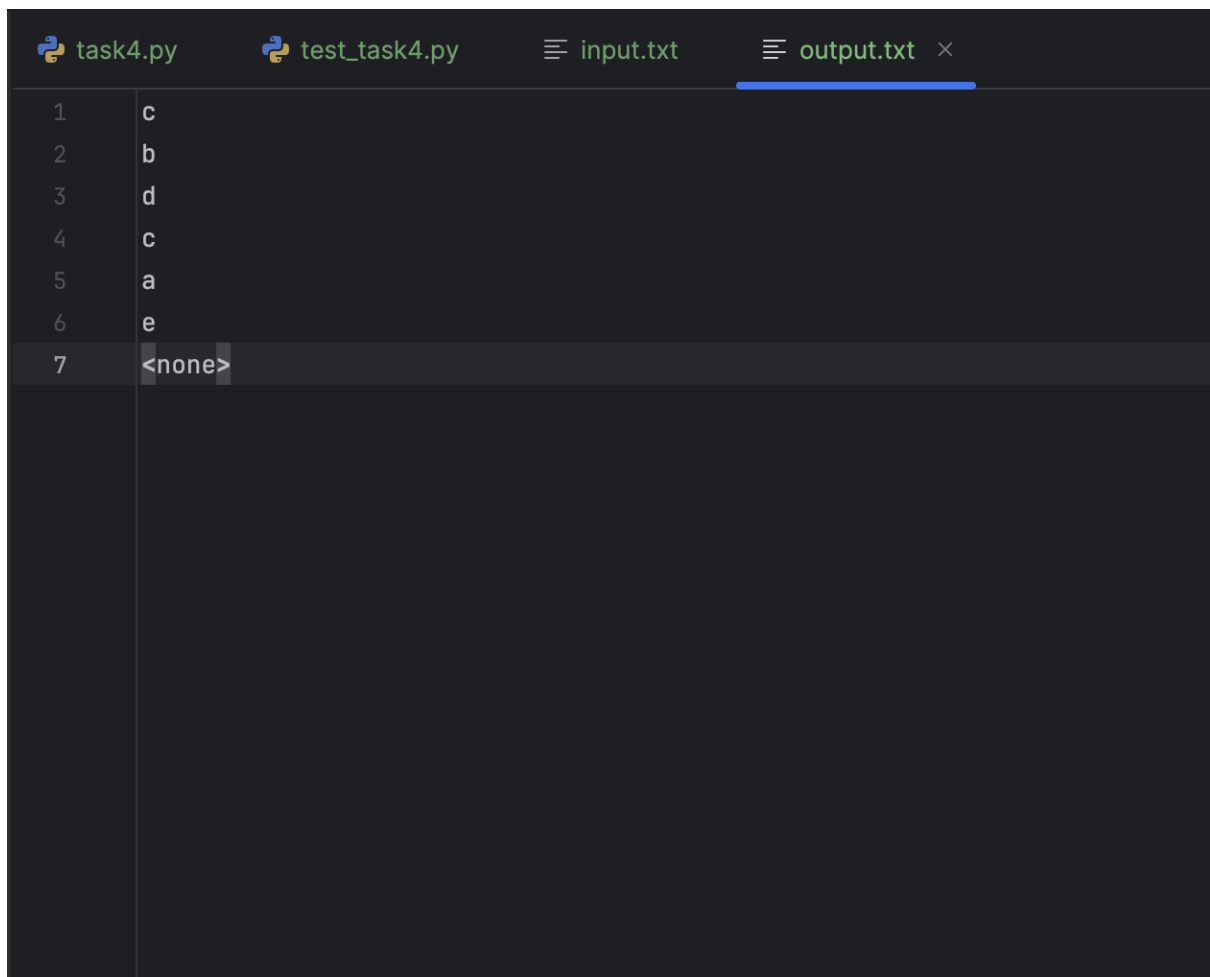
```

Текстовое объяснение решения.

1. Импортируются библиотеки для отслеживания памяти и времени, а также `OrderedDict` для ассоциативного массива.
2. Запускается таймер для измерения времени работы программы.
3. Включается отслеживание памяти.
4. Устанавливаются пути к файлам.
5. Определяется функция `process_commands`, которая выполняет команды над ассоциативным массивом:
 - `put`: добавление или обновление ключа.
 - `get`: получение значения по ключу.
 - `prev`: поиск предыдущего ключа.
 - `next`: поиск следующего ключа.
 - `delete`: удаление ключа.
6. В основном блоке программы:
 - Читаются данные из файла `input.txt`.
 - В первой строке извлекается количество операций `n`.
 - В последующих строках извлекаются команды.
7. Проверяется корректность входных данных (ограничения на количество операций и формат команд).
8. Если данные корректны, выводится информация о задаче и входных данных.
9. Обрабатываются команды с помощью функции `process_commands`.
10. Результаты записываются в файл `output.txt`.
11. Выводится информация о времени работы программы.
12. Выводится информация о затраченной памяти.
13. Останавливается отслеживание памяти.

Результат работы кода на примере из текста задачи:

```
task4.py  test_task4.py  input.txt  output.txt
1  14
2  put zero a
3  put one b
4  put two c
5  put three d
6  put four e
7  get two
8  prev two
9  next two
10 delete one
11 delete three
12 get two
13 prev two
14 next two
15 next four
```



The image shows a code editor with four tabs: `task4.py`, `test_task4.py`, `input.txt`, and `output.txt`. The `output.txt` tab is active. The editor displays a list of seven items, each on a new line, numbered 1 through 7 in the left margin. The items are: 'c', 'b', 'd', 'c', 'a', 'e', and '<none>'. The seventh item, '<none>', is highlighted with a mouse cursor.

Line	Value
1	c
2	b
3	d
4	c
5	a
6	e
7	<none>

Результат работы кода на максимальных и минимальных значениях:

task4.py test_task4.py input.txt × output.txt

1

2

3

1
get two

task4.py test_task4.py input.txt output.txt ×

1

<none>

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004989579992979998 секунд	15341 байт
Пример из задачи	0.0008108750007522758 секунд	16102 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.1820174169843085 секунд	10481917 байт

Вывод по задаче: мною был изучен алгоритм реализации прошитого ассоциативного массива.

Задача №7. Драгоценные камни

Текст задачи.

7 задача. Драгоценные камни

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

Купцы хотят продать шаху n драгоценных камней, которые они привезли с собой. Для этого они выкладывают их перед шахом в ряд, после чего шах оценивает эти камни и принимает решение о том, купит он их или нет. Видов драгоценных камней на Востоке известно не очень много всего 26, поэтому мы будем обозначать виды камней с помощью строчных букв латинского алфавита. Шах обычно оценивает камни следующим образом. Он заранее определил несколько упорядоченных пар типов камней: $(a_1, b_1), (a_2, b_2), \dots, (a_k, b_k)$. 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 $1 \leq i < j \leq n$, а камни S_i и S_j образуют красивую пару, то есть существует такое число $1 \leq q \leq k$, 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 ($1 \leq n \leq 100000$, $1 \leq k \leq 676$) – число камней, которые привезли купцы и число пар, которые шах считает красивыми. Вторая строка входного файла содержит строку S , описывающую типы камней, которые привезли купцы.

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

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

Листинг кода.

```
# Импортируем библиотеки для отслеживания памяти и времени выполнения
программы
import tracemalloc
import time
from lab6.utils import *

# Запускаем таймер для измерения времени работы программы
t_start = time.perf_counter()

# Включаем отслеживание памяти
tracemalloc.start()

# Задаём пути к файлам
current_dir = os.path.dirname(os.path.abspath(__file__)) # Текущая
директория
txtf_dir = os.path.join(os.path.dirname(current_dir), "txtf") # Папка с
файлами
input_path = os.path.join(txtf_dir, "input.txt")

def count_beautiful_pairs(n, k, S, beautiful_pairs):
    """Вычисление количества красивых пар."""
    count = {}
    result = 0

    # Проходим по строке S
    for char in S:
        # Проверяем, образует ли текущий символ красивую пару с ранее
        # встреченными
        for a, b in beautiful_pairs:
            if char == b:
                result += count.get(a, 0)

        # Увеличиваем счётчик для текущего символа
        count[char] = count.get(char, 0) + 1

    return result

# Основной блок программы
if __name__ == "__main__":
    # Читаем данные из файла input.txt
    input_data = open_file(input_path)
    n, k = map(int, input_data[0].split())
    S = input_data[1].strip()
```

```

pairs = input_data[2:] # Читаем пары

# Формируем множество красивых пар
beautiful_pairs = set()
for pair in pairs:
    beautiful_pairs.add((pair[0], pair[1]))

print(beautiful_pairs)

# Проверяем корректность входных данных
if (1 <= n <= 100000) and (1 <= k <= 676) and len(S) == n:
    print(f"\nTask 7\nInput:\n{n} {k}\n{S}\n{beautiful_pairs}")
    delete_prev_values(7)

    # Считаем количество красивых пар
    result = count_beautiful_pairs(n, k, S, beautiful_pairs)

    output_path = get_output_path(7)
    # Записываем результат в файл output.txt
    write_file(str(result), output_path)
    print_output_file(7)
else:
    # Сообщаем об ошибке, если данные некорректны
    print('Введите корректные данные')

# Выводим время работы программы
print("Время работы: %s секунд" % (time.perf_counter() - t_start))
# Выводим количество памяти, затраченной на выполнение программы
print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")

# Останавливаем отслеживание памяти
tracemalloc.stop()

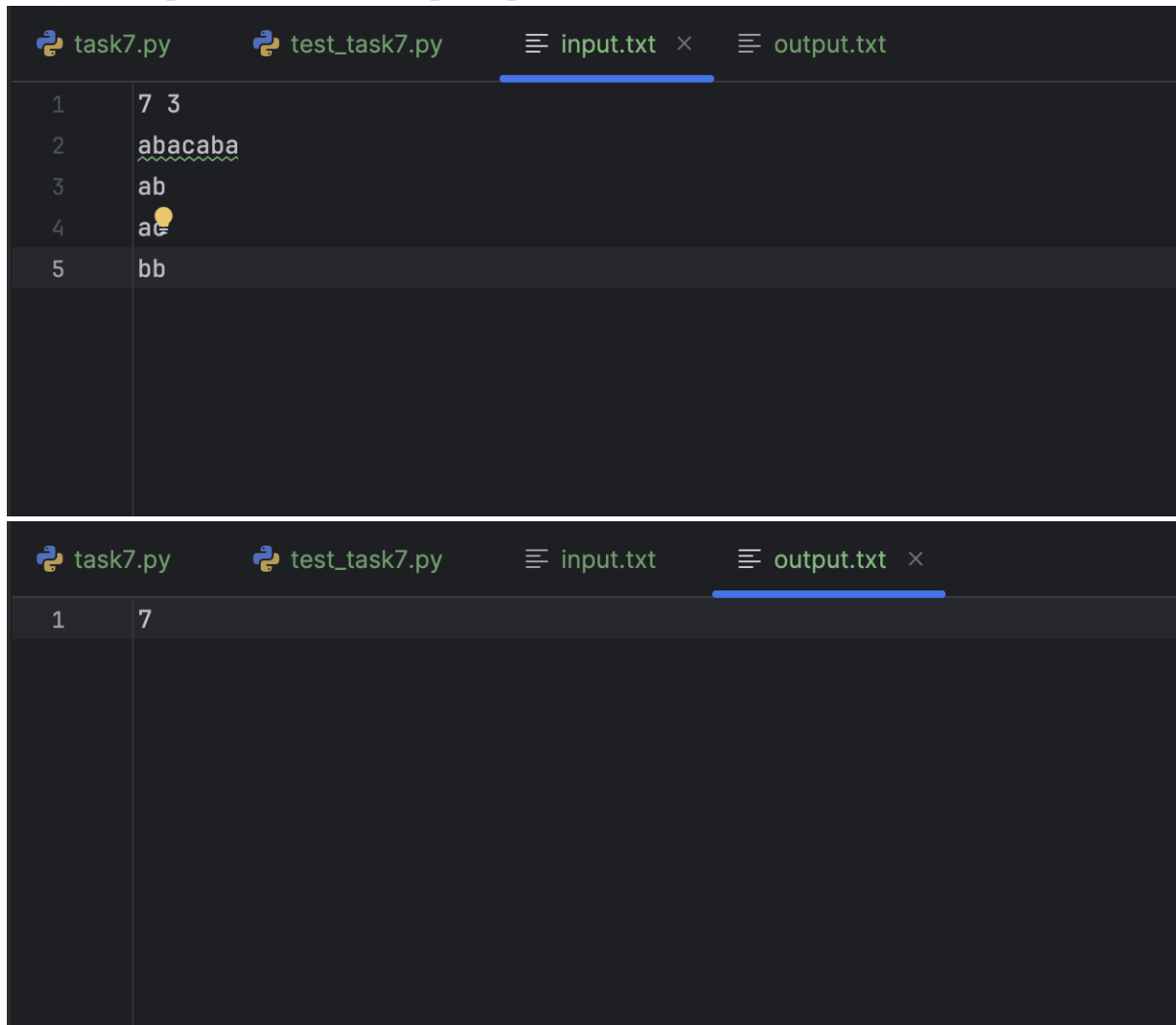
```

Текстовое объяснение решения.

1. Импортируются библиотеки для отслеживания памяти и времени, а также функции из `lab6.utils`.
2. Запускается таймер для измерения времени работы программы.
3. Включается отслеживание памяти.
4. Устанавливаются пути к файлам.
5. Определяется функция `count_beautiful_pairs`, которая:
 - Принимает количество символов, количество пар, строку символов и множество красивых пар.
 - Подсчитывает количество красивых пар в строке, проходя по ней и проверяя, образует ли текущий символ пару с ранее встреченными.
6. В основном блоке программы:
 - Читаются данные из файла `input.txt`.
 - Извлекаются значения `n`, `k`, строка `S` и пары.
7. Формируется множество красивых пар.
8. Проверяется корректность входных данных (ограничения на количество символов, пар и длину строки).
9. Если данные корректны, выводится информация о задаче и входных данных.

10. Вычисляется количество красивых пар с помощью функции `count_beautiful_pairs`.
11. Результат записывается в файл `output.txt`.
12. Выводится информация о времени работы программы.
13. Выводится информация о затраченной памяти.
14. Останавливается отслеживание памяти.

Результат работы кода на примере из текста задачи:



The image shows two screenshots of a code editor interface. The top screenshot shows the 'input.txt' file with the following content:

```
1 7 3
2 abacaba
3 ab
4 ac
5 bb
```

The bottom screenshot shows the 'output.txt' file with the following content:

```
1 7
```


Результат работы кода на максимальных и минимальных значениях:

```
task7.py test_task7.py input.txt x output.txt
1 1 1
2 a
3 ab
```

```
task7.py test_task7.py input.txt output.txt x
1 0
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005294170005072374 секунд	15412 байт
Пример из задачи	0.0008986670000012964 секунд	15506 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.19008412497350946 секунд	11521614 байт

Вывод по задаче: мною был изучен алгоритм поиска “красивых пар”.

Вывод

В ходе лабораторной работы я узнал про хеширование и хеш-таблицы.