САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №6

по курсу «Алгоритмы и структуры данных»

Вариант 1

Выполнил:

Азаренков Георгий Денисович

К34421

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[Содержание отчета 2](#_Toc179016445)

[Комментарий к общему коду 3](#_Toc179016446)

[Задача 1 5](#_Toc179016447)

[Текст задачи 5](#_Toc179016448)

[Код решения 5](#_Toc179016449)

[Результаты выполнения 6](#_Toc179016450)

[Задача 2 7](#_Toc179016451)

[Текст задачи 7](#_Toc179016452)

[Код решения 7](#_Toc179016453)

[Результаты выполнения 8](#_Toc179016454)

[Задача 4 10](#_Toc179016455)

[Текст задачи 10](#_Toc179016456)

[Код решения 11](#_Toc179016457)

[Результаты выполнения 13](#_Toc179016458)

[Вывод 14](#_Toc179016459)

# Комментарий к общему коду

При использовании всех задач я использую код из файла “common.py”, который находится в корне репозитория.

from typing import List, Callable  
  
  
def solve(solver: Callable[[List[str]], List[str]]) -> None:  
 with open("../../input.txt", "r") as file:  
 lines = file.readlines()  
 input\_lines = list(map(lambda x: x.strip(), lines))  
  
 output\_lines = solver(input\_lines)  
  
 with open("../../output.txt", "w") as file:  
 for line in output\_lines:  
 file.write(f"{line}\n")

Использование функции “solve” позволило мне избавиться от необходимости раз за разом имплементировать логику чтения и сохранения данных.

Номера заданий после расшифровки получились такие

H(1) = (21 ⋅ 1 mod 19) mod 9  
  
Ответ: номер задачи H(1) = 3.

# Задача 1

## Текст задачи

A paper with text and numbers

Description automatically generated

## Код решения

from common import solve  
  
  
def get\_solution(input\_lines: list[str]) -> list[str]:  
 *"""  
 Задача: Реализовать множество с операциями добавления, удаления и проверки существования элемента.  
 На вход поступает список строк с командами, на выходе список ответов для операций проверки (команда '?').  
 """* key\_set = set()  
  
 result = []  
  
 for line in input\_lines[1:]:  
  
 operation, value\_str = line.split()  
  
 value = int(value\_str)  
  
 if operation == "A":  
 key\_set.add(value)  
  
 elif operation == "D":  
 key\_set.discard(value)  
  
 elif operation == "?":  
 if value in key\_set:  
 result.append("Y")  
 else:  
 result.append("N")  
  
 return result  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A screenshot of a quiz

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

# Задача 2

## Текст задачи

A paper with text and numbers

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_data: List[str]) -> List[str]:  
 *"""  
 Реализует менеджер телефонной книги, который обрабатывает команды добавления, удаления и поиска номеров.  
 """* phone\_book = {}  
 results = []  
  
 n = int(input\_data[0])  
  
 for i in range(1, n + 1):  
 parts = input\_data[i].split()  
 command = parts[0]  
  
 if command == "add":  
 number = parts[1]  
 name = parts[2]  
  
 phone\_book[number] = name  
  
 elif command == "del":  
 number = parts[1]  
  
 phone\_book.pop(number, None)  
  
 elif command == "find":  
 number = parts[1]  
  
 result = phone\_book.get(number, "not found")  
 results.append(result)  
  
 return results  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

# Задача 4

## Текст задачи

A screenshot of a cell phone

Description automatically generated

## Код решения

from typing import List, Dict, Optional  
  
from common import solve  
  
NONE\_VALUE = "<none>"  
  
  
class ListNode:  
 *"""  
 Класс узла для связанного списка  
 """* def \_\_init\_\_(self, key: str, value: str):  
 self.key: str = key  
 self.value: str = value  
 self.prev: Optional[ListNode] = None  
 self.next: Optional[ListNode] = None  
  
  
class AssociativeArray:  
 *"""  
 Класс ассоциативного массива с поддержкой порядка вставки  
 """* def \_\_init\_\_(self):  
 self.map: Dict[str, ListNode] = {}  
 self.head: Optional[ListNode] = None  
 self.tail: Optional[ListNode] = None  
  
 def put(self, key: str, value: str) -> None:  
 if key in self.map:  
 *# Обновляем значение существующего ключа* self.map[key].value = value  
 else:  
 *# Создаем новый узел и добавляем в конец списка* new\_node = ListNode(key, value)  
  
 if not self.head:  
 self.head = self.tail = new\_node  
 else:  
 self.tail.next = new\_node  
 new\_node.prev = self.tail  
 self.tail = new\_node  
  
 self.map[key] = new\_node  
  
 def get(self, key: str) -> str:  
 if key in self.map:  
 return self.map[key].value  
  
 return NONE\_VALUE  
  
 def delete(self, key: str) -> None:  
 if key not in self.map:  
 return  
  
 node = self.map.pop(key)  
  
 *# Удаляем узел из связанного списка* if node.prev:  
 node.prev.next = node.next  
 else:  
 self.head = node.next  
  
 if node.next:  
 node.next.prev = node.prev  
 else:  
 self.tail = node.prev  
  
 def prev(self, key: str) -> str:  
 if key not in self.map:  
 return NONE\_VALUE  
  
 node = self.map[key]  
  
 if node.prev:  
 return node.prev.value  
  
 return NONE\_VALUE  
  
 def next(self, key: str) -> str:  
 if key not in self.map:  
 return NONE\_VALUE  
  
 node = self.map[key]  
  
 if node.next:  
 return node.next.value  
  
 return NONE\_VALUE  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Реализует ассоциативный массив с операциями get, prev, next, put, delete.  
 Обрабатывает список команд и возвращает результаты операций get, prev, next.  
 """* n = int(input\_lines[0])  
 aa = AssociativeArray()  
 output = []  
  
 for i in range(1, n + 1):  
 parts = input\_lines[i].split()  
 command = parts[0]  
  
 if command == "put":  
 key = parts[1]  
 value = parts[2]  
 aa.put(key, value)  
  
 elif command == "get":  
 key = parts[1]  
 result = aa.get(key)  
 output.append(result)  
  
 elif command == "delete":  
 key = parts[1]  
 aa.delete(key)  
  
 elif command == "prev":  
 key = parts[1]  
 result = aa.prev(key)  
 output.append(result)  
  
 elif command == "next":  
 key = parts[1]  
 result = aa.next(key)  
 output.append(result)  
  
 return output  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

# Вывод

В ходе данной лабораторной работы я научился решать задачи по теме. Написанные программы были протестированы с измерением затрат времени и оперативной памяти. Все программы работаю корректно и укладываются в установленные ограничения.