САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №6
по курсу «Алгоритмы и структуры данных»
Хеширование. Хеш-таблицы

Вариант 15

Выполнил:
Левахин Лев Александрович
К3140

Проверил:
--

Санкт-Петербург
2024 г.

Содержание отчета

Задания по варианту: 2, 3, 5

Задача 2 - Телефонная книга

Задача 3 – Хеширование с цепочками

Задача 5 – Выборы в США

Задания по выбору: 6, 8

Задача 6 - Фибоначчи возвращается

Задача 8 - Почти интерактивная хеш-таблица

Задачи по варианту

Задача №2. Телефонная книга

Листинг кода.

```
from lab6 import utils
import os

CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


class PhoneBook:
    """Телефонная книга"""

    def __init__(self):
        self.contacts_book = {}

    def __str__(self):
        all_contacts = ""
        for number in self.contacts_book:
            all_contacts += f"{number} {self.contacts_book[number]}"
        return all_contacts

    def add_contact(self, number:str, name: str):
        self.contacts_book[number] = name

    def del_contact(self, number: str):
        self.contacts_book.pop(number, None)

    def find_number(self, number: str):
        if number in self.contacts_book.keys():
            return self.contacts_book[number]
        return "not found"

    def commands_to_actions(self, commands:list[str]):
        result = []
        for command in commands:
            command = command.split()
            operation = command[0]
            match operation:
                case "add":
                    number, name = command[1], command[2]
                    self.add_contact(number, name)
                case "del":
                    number = command[1]
                    self.del_contact(number)
                case "find":
                    number = command[1]
                    result.append(self.find_number(number))
                case _:
                    return f"Команды {operation} нет!"

        return result


def input_data():
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    n, commands = data[0], data[1:]
    return n, commands


def main():
    n, commands = input_data()
```

```
    my_phone_book = PhoneBook()
    result = my_phone_book.commands_to_actions(commands)

    return result


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, result)
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab6.task2.src.task2 import *
from lab6 import utils
import unittest
import datetime
import tracemalloc


class TaskTest2(unittest.TestCase):

    def test_func_performance(self):
        """Тест функции на время и память"""
        # given
        my_phone_book = PhoneBook()
        contacts = ["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"]

        max_allowed_time = datetime.timedelta(seconds=6)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = my_phone_book.commands_to_actions(contacts)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, ['Mom', 'not found', 'police', 'not
found', 'Mom', 'daddy'])
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 512)


    def test_func_correctly(self):
        """Тест на корректность работы"""
```

```
        # given
        contacts1 = ["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"]
        contacts2 = ["find 3839442","add 123456 me","add 0 granny","find
0","find 123456","del 0","del 0","find 0"]
        phone_book1 = PhoneBook()
        phone_book2 = PhoneBook()

        # when
        result1 = phone_book1.commands_to_actions(contacts1)
        result2 = phone_book2.commands_to_actions(contacts2)

        # then
        self.assertEqual(result1, ['Mom', 'not found', 'police', 'not
found', 'Mom', 'daddy'])
        self.assertEqual(result2, ['not found', 'granny', 'me', 'not
found'])


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

Скрины работы тестов:

Задача №3.

Листинг кода.

```
from lab6 import utils
import os

CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def hash_function(s, p=1000000007, x=263):
    hash_value = 0
    for i, char in enumerate(s):
        hash_value = (hash_value + ord(char) * (x ** i)) % p
    return hash_value


def commands_to_actions(m: int, n: int, requests: list[str]) -> list:
    hash_table = [[] for i in range(m)]
    result = []
    for query in requests:
        query = query.split()
        command = query[0]
        match command:
            case "add":
                string = query[1]
                hash_val = hash_function(string) % m
                if string not in hash_table[hash_val]:
                    hash_table[hash_val].insert(0, string)

            case "del":
                string = query[1]
                hash_val = hash_function(string) % m
                if string in hash_table[hash_val]:
                    hash_table[hash_val].remove(string)
```

```
            case "find":
                string = query[1]
                hash_val = hash_function(string) % m
                if string in hash_table[hash_val]:
                    result.append("yes")
                else:
                    result.append("no")

            case "check":
                i = int(query[1])
                if hash_table[i]:
                    result.append(" ".join(hash_table[i]))
                else:
                    result.append("")

            case _:
                return f"Команды {command} не существует!"

    return result


def input_data():
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    m, n, requests = int(data[0]), int(data[1]), data[2:]
    return m, n, requests


def main():
    m, n, requests = input_data()
    result = commands_to_actions(m, n, requests)
    return result


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, result, True)
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab6.task3.src.task3 import *
from lab6 import utils
import unittest
import datetime
import tracemalloc


class TaskTest3(unittest.TestCase):

    def test_func_performance(self):
        """Тест функции на время и память"""
        # given
        m, n, requests = 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']
```

```
        max_allowed_time = datetime.timedelta(seconds=7)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = commands_to_actions(m, n, requests)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, ['', 'no', 'yes', 'yes', 'no', '', 'add
help', ''])
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 512)


    def test_func_correctly(self):
        """Тест на корректность работы"""
        # given
        m1, n1, requests1 = 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']
        m2, n2, requests2 = 4, 8, ['add test', 'add test', 'find test',
'del test', 'find test', 'find Test', 'add Test', 'find Test']
        m3, n3, requests3 = 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']

        # when
        result1 = commands_to_actions(m1, n1, requests1)
        result2 = commands_to_actions(m2, n2, requests2)
        result3 = commands_to_actions(m3, n3, requests3)

        # then
        self.assertEqual(result1, ['HellO world', 'no', 'yes', 'HellO',
'GooD luck'])
        self.assertEqual(result2, ['yes', 'no', 'no', 'yes'])
        self.assertEqual(result3, ['', 'no', 'yes', 'yes', 'no', '', 'add
help', ''])


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

Скрины работы тестов:

Задача 5. Выборы в США

Листинг кода.

```
from lab6 import utils
import os
```

```
CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def calculate_election_results(votes_list):
    votes = {}

    for entry in votes_list:
        candidate, count = entry.split()
        count = int(count)
        if candidate in votes:
            votes[candidate] += count
        else:
            votes[candidate] = count

    sorted_candidates = sorted(votes.items())

    result = [f"{candidate} {total_votes}" for candidate, total_votes in
sorted_candidates]
    return result


def input_data():
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    return data


def main():
    candidates_list = input_data()
    result = calculate_election_results(candidates_list)
    return result


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, result)
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab6.task5.src.task5 import *
from lab6 import utils
import unittest
import datetime
import tracemalloc


class TaskTest5(unittest.TestCase):

    def test_func_performance(self):
        """Тест функции на время и память"""
        # given
        candidates_list = ['McCain 10', 'McCain 5', 'Obama 9', 'Obama 8',
'McCain 1']

        max_allowed_time = datetime.timedelta(seconds=2)   # Задаю
```

```
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = calculate_election_results(candidates_list)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, ['McCain 16', 'Obama 17'])
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 256)

    def test_func_correctly(self):
        """Тест на корректность работы"""
        # given
        candidates_list1 = ['McCain 10', 'McCain 5', 'Obama 9', 'Obama 8',
'McCain 1']
        candidates_list2 = ['ivanov 100', 'ivanov 500', 'ivanov 300', 'petr
70', 'tourist 1', 'tourist 2']
        candidates_list3 = ['bur 1']

        # when
        result1 = calculate_election_results(candidates_list1)
        result2 = calculate_election_results(candidates_list2)
        result3 = calculate_election_results(candidates_list3)

        # then
        self.assertEqual(result1, ['McCain 16', 'Obama 17'])
        self.assertEqual(result2, ['ivanov 900', 'petr 70', 'tourist 3'])
        self.assertEqual(result3, ['bur 1'])


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

Скрины работы тестов:

Дополнительные задачи

Задача №6. Фиббоначи возвращается

Листинг кода.

```
from lab6 import utils
import os

CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def is_fibonacci(num_str, fib_set):
    """
    Функция генерирует числа Фибоначчи и добавляет их в множество,
    пока длина числа не превысит заданное число цифр
    Множество обеспечивает эффективный поиск
    """
    return num_str in fib_set


def generate_fibonacci_until(max_digits=1000):
    """Функция проверяет, является ли строковое представление числа
Фибоначчи."""
    fib_set = set()
    a, b = 1, 1
    fib_set.add(str(a))
    fib_set.add(str(b))

    while True:
        a, b = b, a + b
        fib_str = str(b)
        if len(fib_str) > max_digits:
            break
        fib_set.add(fib_str)

    return fib_set


def is_fib_list(lst: list):
    result = []
    fib_set = generate_fibonacci_until()
    for query in lst:
        if is_fibonacci(query, fib_set):
            result.append("Yes")
        else:
            result.append("No")

    return result


def input_data():
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    n, fib_numbers = int(data[0]), data[1:]
    return n, fib_numbers


def main():
    n, fib_numbers = input_data()
    result = is_fib_list(fib_numbers)
    return result


if __name__ == "__main__":
```

```
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, result, True)
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab6.task6.src.task6 import *
from lab6 import utils
import unittest
import datetime
import tracemalloc


class TaskTest6(unittest.TestCase):

    def test_func_performance(self):
        """Тест функции на время и память"""
        # given
        fib_numbers = ["1", "2", "3", "4", "5", "6", "7", "8"]

        max_allowed_time = datetime.timedelta(seconds=2)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = is_fib_list(fib_numbers)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, ['Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
'No', 'Yes'])
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 128)


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

Скрины работы тестов:

Задача №8. Почти полная хеш-таблица

Листинг кода.

```
from lab6 import utils
import os

CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def hash_table_operations(n, x, a, b, ac, bc, ad, bd) -> list:
    hash_table = set()

    for i in range(n):
        if x in hash_table:
            a = (a + ac) % 10 ** 3
            b = (b + bc) % 10 ** 15
        else:
            hash_table.add(x)
            a = (a + ad) % 10 ** 3
            b = (b + bd) % 10 ** 15

        x = (x * a + b) % 10**15

    return [x, a, b]


def input_data():
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    line1 = utils.str_to_list(data[0])
    line2 = utils.str_to_list(data[1])
    line1 = [int(i) for i in line1]
    line2 = [int(i) for i in line2]
    n, x, a, b = line1[0], line1[1], line1[2], line1[3]
    ac, bc, ad, bd = line2[0], line2[1], line2[2], line2[3]

    return n, x, a, b, ac, bc, ad, bd


def main():
    n, x, a, b, ac, bc, ad, bd = input_data()
    result = hash_table_operations(n, x, a, b, ac, bc, ad, bd)
    return result


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, [result])
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:

Листинг кода:

```
from lab6.task8.src.task8 import *
from lab6 import utils
```

```
import datetime
import tracemalloc
import unittest


class TaskTest8(unittest.TestCase):

    def test_func_performance(self):
        """Тест функции на время и память"""
        # given
        n, x, a, b, ac, bc, ad, bd = 4, 0, 0, 0, 1, 1, 0, 0

        max_allowed_time = datetime.timedelta(seconds=5)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = hash_table_operations(n, x, a, b, ac, bc, ad, bd)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, [3, 1, 1])
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 256)


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

Скрины работы тестов:

Вывод

Работа с хешированием требует довольно много внимательности и усилий. Реализация хеш-таблиц и списков помогает в реализации сложных алгоритмов.