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

Отчет по лабораторной работе №5
по курсу «Алгоритмы и структуры данных»
Тема: Стек, очередь, связанный список.
Вариант 3

Выполнил:
Бай М.О.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Оглавление

Задачи по варианту

Задача №1. Множество

Текст задачи:

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}.

Код программы:

```
import os
import sys

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '../..')))
from Lab6.utils import read_input, write_output, decorate  # Adjust this import if necessary


class CustomSet:
    def __init__(self):
        self.data = {}

    def add(self, x):
        """Добавление элемента в множество."""
        self.data[x] = True

    def remove(self, x):
        """Удаление элемента из множества."""
        if x in self.data:
            del self.data[x]

    def exists(self, x):
        """Проверка существования элемента в множестве."""
        return x in self.data


def main():
    # Читаем входные данные
    n = int(read_input(1)[0])  # Читаем количество операций
    operations = read_input(1)[1:n + 1]  # Читаем операции, начиная со второй строки
```

```
    custom_set = CustomSet()
    results = []

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

        if op_type == 'A':
            custom_set.add(x)
        elif op_type == 'D':
            custom_set.remove(x)
        elif op_type == '?':
            results.append('Y' if custom_set.exists(x) else 'N')

    write_output(1, *results)  # Записываем результаты в выходной файл
    print('\n'.join(results) + '\n')


if __name__ == '__main__':
    decorate(task=1, task_name='Array')  # Передаем номер задачи и название модуля
```

Описание работы программы:

Основные Компоненты:

Импорты:

Код использует модули os и sys для работы с файловой системой и для добавления путей к модулям. Также импортируются функции read_input, write_output и decorate из Lab6.utils, которые используются для чтения и записи данных, а также для логирования.

Класс CustomSet:

Инициализация:

В конструкторе __init__ создается пустой словарь self.data, который будет использоваться для хранения элементов множества.

Метод add(x):

Добавляет элемент x в множество. Используется словарь, чтобы обеспечить уникальность элементов (ключи словаря не могут повторяться).

Метод remove(x):

Удаляет элемент x из множества, если он существует. Если элемент отсутствует, ничего не происходит.

Метод exists(x):

Проверяет, существует ли элемент x в множестве. Возвращает True, если элемент есть, и False в противном случае.

Функция main():

Считывает количество операций n и сами операции из входного файла.

Создает экземпляр CustomSet и инициализирует список results для хранения результатов операций проверки.

Для каждой операции:
Разделяет строку операции на тип (op_type) и значение (x).
В зависимости от типа операции (A — добавление, D — удаление, ? — проверка существования) вызывает соответствующие методы класса CustomSet.
Для операций проверки добавляет результат (Y или N) в список results.
Записывает результаты в выходной файл и выводит их на экран.
Точка Входа:
Проверяет, запущен ли код как основной модуль, и вызывает функцию decorate, передавая номер задачи и название модуля для логирования.

Тесты:

```
import unittest
from Lab6.task1.src.Array import *

class TestCustomSet(unittest.TestCase):

    def setUp(self):
        """Создание нового экземпляра CustomSet перед каждым тестом."""
        self.custom_set = CustomSet()

    def test_should_add_element(self):
        """Тестирование добавления элементов в множество."""
        self.custom_set.add(5)
        self.assertTrue(self.custom_set.exists(5), "Element 5 should exist after adding.")

        self.custom_set.add(10)
        self.assertTrue(self.custom_set.exists(10), "Element 10 should exist after adding.")

    def test_should_remove_element(self):
        """Тестирование удаления элементов из множества."""
        self.custom_set.add(5)
        self.custom_set.remove(5)
        self.assertFalse(self.custom_set.exists(5), "Element 5 should not exist after removing.")

    def test_should_exists_element(self):
        """Тестирование проверки существования элементов в множестве."""
        self.custom_set.add(25)
        self.assertTrue(self.custom_set.exists(25), "Element 25 should exist after adding.")

        self.assertFalse(self.custom_set.exists(30), "Element 30 should not exist if not added.")


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

Вывод по задаче:
Реализация пользовательского множества в данном коде является хорошей основой для работы с задачами, связанными с множествами и операциями над ними.

Задача №2. Телефонная книга

Текст задачи:

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

- `add number name` – это команда означает, что пользователь добавляет в телефонную книгу человека с именем `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.

Код программы:

```
import sys
import os

from Lab6.utils import read_input, write_output, decorate

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '../..')))

class PhoneBook:
    def __init__(self):
        self.input_file = read_input(2)
        self.n = self.input_file[0]
        self.array = []
        [self.array.append(i.split()) for i in self.input_file[1:]]
        self.phone_book = []

    def add(self, number, name):
        for i in self.phone_book:
            if i[0] == number:
                self.dell(number)
        self.phone_book.append((number, name))

    def dell(self, number):
        for i in self.phone_book:
            if i[0] == number:
                self.phone_book.remove(i)

    def find(self, number):
        for i in self.phone_book:
            if i[0] == number:
                return i[1]
        return 'not found'

def main():
    book = PhoneBook()
    res = []
    for row in book.array:
        if row[0] == 'add':
```

```
            book.add(row[1], row[2])
        elif row[0] == 'find':
            res.append(book.find(row[1]))
        elif row[0] == 'del':
            book.dell(row[1])

    write_output(2, *res)
    [print(i) for i in res]
    print()

if __name__ == '__main__':
    decorat
```

Описание работы программы:

Основные Компоненты:

Импорты:

Импортируются модули sys и os для работы с файловой системой и управления путями к модулям. Также импортируются функции read_input, write_output и decorate из Lab6.utils, которые используются для чтения и записи данных, а также для логирования.

Класс PhoneBook:

Инициализация:

В конструкторе __init__ считываются данные из входного файла. Первая строка содержит количество операций, а остальные строки содержат сами операции. Каждая операция разбивается на части и добавляется в список self.array.

Создается пустой список self.phone_book, который будет использоваться для хранения записей (номер, имя).

Метод add(number, name):

Добавляет запись в телефонную книгу. Если номер уже существует, вызывается метод dell(number) для его удаления перед добавлением новой записи.

Метод dell(number):

Удаляет запись с указанным номером из телефонной книги. Если номер найден, он удаляется из списка self.phone_book.

Метод find(number):

Находит имя по указанному номеру. Если номер найден, возвращает имя, иначе возвращает строку 'not found'.

Функция main():

Создает экземпляр PhoneBook и инициализирует список res для хранения результатов операций поиска.

Для каждой операции в book.array:

Если операция — add, вызывается метод add.
Если операция — find, результат поиска добавляется в список res.
Если операция — del, вызывается метод dell.
Результаты поиска записываются в выходной файл и выводятся на экран.
Точка Входа:
Проверяет, запущен ли код как основной модуль, и вызывает функцию decorate
Тесты:

```
import unittest
from Lab6.task1.src.Array import *

class TestCustomSet(unittest.TestCase):

    def setUp(self):
        """Создание нового экземпляра CustomSet перед каждым тестом."""
        self.custom_set = CustomSet()

    def test_should_add_element(self):
        """Тестирование добавления элементов в множество."""
        self.custom_set.add(5)
        self.assertTrue(self.custom_set.exists(5), "Element 5 should exist after adding.")

        self.custom_set.add(10)
        self.assertTrue(self.custom_set.exists(10), "Element 10 should exist after adding.")

    def test_should_remove_element(self):
        """Тестирование удаления элементов из множества."""
        self.custom_set.add(5)
        self.custom_set.remove(5)
        self.assertFalse(self.custom_set.exists(5), "Element 5 should not exist after removing.")

    def test_should_exists_element(self):
        """Тестирование проверки существования элементов в множестве."""
        self.custom_set.add(25)
        self.assertTrue(self.custom_set.exists(25), "Element 25 should exist after adding.")

        self.assertFalse(self.custom_set.exists(30), "Element 30 should not exist if not added.")


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

Вывод по задаче:
Код предоставляет базовую реализацию телефонной книги с поддержкой операций добавления, удаления и поиска.

Задача №4. Прошитый ассоциативный массив

Текст задачи:

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.

Код программы:

```
import sys
import os

from Lab6.utils import read_input, write_output, decorate

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '../..')))

class AssociativeArray:
    def __init__(self):
        self.input_file = read_input(4)
        self.n = self.input_file[0]
        self.array = []
        [self.array.append(i.split()) for i in self.input_file[1:]]
        self.ass_array = {}

    def put(self, key, value):
        self.ass_array[key] = value

    def get(self, key):
        for i in self.ass_array.keys():
            if i == key:
                return self.ass_array[i]
        return '<None>'
```

```
    def prev(self, key):
        if key in self.ass_array:
            keys = list(self.ass_array.keys())
            idx = keys.index(key)
            if idx > 0:
                return self.ass_array[keys[idx - 1]]
            else:
                return "<none>"
        else:
            return "<none>"

    def next(self, key):
        if key in self.ass_array:
            keys = list(self.ass_array.keys())
            idx = keys.index(key)
            if idx < len(keys) - 1:
                return self.ass_array[keys[idx + 1]]
            else:
                return "<none>"
        else:
            return "<none>"

    def delete(self, key):
        del self.ass_array[key]

def main():
    ass = AssociativeArray()
    res = []
    for i in ass.array:
        if i[0] == 'put':
            ass.put(i[1], i[2])
        elif i[0] == 'get':
            res.append(ass.get(i[1]))
        elif i[0] == 'prev':
            res.append(ass.prev(i[1]))
        elif i[0] == 'next':
            res.append(ass.next(i[1]))
        elif i[0] == 'delete':
            ass.delete(i[1])

    write_output(4, *res)
    [print(i) for i in res]
    print()

if __name__ == '__main__':
    decorate(task=4, task_name='AssociativeArray')
```

Описание работы программы:

Основные Компоненты:

Импорты:

Импортируются модули sys и os для работы с файловой системой. Также импортируются функции read_input, write_output и decorate из Lab6.utils для работы с вводом/выводом и логированием.

Класс AssociativeArray:

Инициализация:

В конструкторе __init__ считываются данные из входного файла. Первая строка содержит количество операций, а остальные строки содержат команды. Каждая команда разбивается на части и добавляется в список self.array.
Создается пустой словарь self.ass_array для хранения пар ключ-значение.
Метод put(key, value):
Добавляет или обновляет запись в ассоциативном массиве по заданному ключу.
Метод get(key):
Возвращает значение по указанному ключу. Если ключ не найден, возвращает строку <None>.
Метод prev(key):
Возвращает значение предыдущего ключа относительно указанного. Если ключ не найден или это первый ключ, возвращает строку <none>.
Метод next(key):
Возвращает значение следующего ключа относительно указанного. Если ключ не найден или это последний ключ, возвращает строку <none>.
Метод delete(key):
Удаляет запись с указанным ключом из ассоциативного массива.
Функция main():
Создает экземпляр AssociativeArray и инициализирует список res для хранения результатов операций.
Для каждой операции в ass.array:
Если операция — put, добавляется пара ключ-значение.
Если операция — get, результат поиска добавляется в список res.
Если операция — prev, добавляется значение предыдущего ключа.
Если операция — next, добавляется значение следующего ключа.
Если операция — delete, удаляется запись по ключу.
Результаты операций записываются в выходной файл и выводятся на экран.
Точка Входа:
Проверяет, запущен ли код как основной модуль, и вызывает функцию decorate, передавая параметры задачи.

Тесты

```
import unittest
from Lab6.task1.src.Array import *

class TestCustomSet(unittest.TestCase):

    def setUp(self):
        """Создание нового экземпляра CustomSet перед каждым тестом."""
        self.custom_set = CustomSet()
```

```
    def test_should_add_element(self):
        """Тестирование добавления элементов в множество."""
        self.custom_set.add(5)
        self.assertTrue(self.custom_set.exists(5), "Element 5 should exist after adding.")

        self.custom_set.add(10)
        self.assertTrue(self.custom_set.exists(10), "Element 10 should exist after adding.")

    def test_should_remove_element(self):
        """Тестирование удаления элементов из множества."""
        self.custom_set.add(5)
        self.custom_set.remove(5)
        self.assertFalse(self.custom_set.exists(5), "Element 5 should not exist after removing.")

    def test_should_exists_element(self):
        """Тестирование проверки существования элементов в множестве."""
        self.custom_set.add(25)
        self.assertTrue(self.custom_set.exists(25), "Element 25 should exist after adding.")

        self.assertFalse(self.custom_set.exists(30), "Element 30 should not exist if not added.")


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

Вывод по задаче:

Код предоставляет базовую реализацию ассоциативного массива с поддержкой операций добавления, поиска, удаления, а также получения предыдущего и следующего элементов.

Задача №5. Выборы в США

Текст задачи:

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

Код программы:

```
import sys
import os

from Lab6.utils import read_input, write_output, decorate
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '../..')))

def process_elections(rd):
    votes = {}
```

```
    infile = rd
    for line in infile:
        parts = line.strip().split()
        candidate = parts[0]
        num_votes = int(parts[1])

        # Добавляем голоса кандидату
        if candidate in votes:
            votes[candidate] += num_votes
        else:
            votes[candidate] = num_votes


    sorted_candidates = sorted(votes.items())

    out = ''
    for candidate, total_votes in sorted_candidates:
        out = out + f"{candidate} {total_votes}\n"
    return out

def main():
    out = process_elections(read_input(5))
    write_output(5, out)
    print(out)
    print()


if __name__ == '__main__':
    decorate(task=5, task_name='Process_elections')
```

Описание работы кода:

Основные Компоненты:

Импорты:

Импортируются модули sys и os для работы с файловой системой.

Импортируются функции read_input, write_output и decorate из Lab6.utils, которые используются для работы с вводом/выводом и логированием.

Функция process_elections(rd):

Принимает на вход объект rd, содержащий строки с данными о голосах.

Создается словарь votes, где ключами являются имена кандидатов, а значениями — количество голосов, полученных каждым кандидатом.

Для каждой строки в infile (входные данные):

Строка разбивается на части, где первая часть — это имя кандидата, а вторая — количество голосов.

Если кандидат уже есть в словаре votes, количество голосов обновляется. В противном случае добавляется новая запись.

После обработки всех строк данные сортируются по именам кандидатов.

Результаты форматируются в строку out, где каждый кандидат и его общее количество голосов выводятся в формате "{candidate} {total_votes}\n".

Функция main():

Вызывает функцию process_elections, передавая ей результат работы read_input(5), который считывает входные данные.

Результаты записываются в выходной файл с помощью функции write_output(5, out).

Результаты также выводятся на экран.

Точка Входа:

Проверяет, запущен ли код как основной модуль, и вызывает функцию decorate, передавая параметры задачи.

Тесты

```
from Lab6.task5.src.Process_elections import *

import unittest


class TestProcessElections(unittest.TestCase):


    def test_should_no_candidates(self):
        input_data = []
        expected_output = ""
        self.assertEqual(process_elections(input_data), expected_output)

    def test_should_invalid_vote_count(self):
        input_data = ["Alice ten"]
        with self.assertRaises(ValueError):
            process_elections(input_data)

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

Вывод по задаче:

Данный код реализует программу для обработки результатов выборов, считывая данные о голосах за кандидатов и выводя итоговые результаты.

Вывод

В данной работе мы изучили работу со стеками, очередями и списками.