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

Отчет по лабораторной работе №4
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. Стек

Текст задачи:

Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо "+ N", либо "–". Команда "+ N"означает добавление в стек числа N, по модулю не превышающего 10^9. Команда "–"означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выполнения команд не превысит 10^6 элементов.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится M $(1 \leq M \leq 10^6)$ – число команд. Каждая последующая строка исходного файла содержит ровно одну команду.
- **Формат выходного файла (output.txt).** Выведите числа, которые удаляются из стека с помощью команды "–", по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из стека. Гарантируется, что изъятий из пустого стека не производится.

Код программы:

```
import sys
import os

from Lab4.utils import read_input, write_output, decorate

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

class Stack:
    def __init__(self):
        self.input_file = read_input(1)
        self.array = self.input_file[1:]
        self.stack = []

    def pop(self):
        removed = self.stack.pop()
        return removed

    def push(self, item):
        self.stack.append(item)


    def result(self):
        pops = []
        for i in self.array:
            arr = i.split()
            if arr[0] == '+':
                self.push(arr[1])
            elif arr[0] == '-':
                pops.append(str(self.pop()))
        return pops
```

```
def main():
    stc = Stack()
    pops = stc.result()
    write_output(1, *pops)
    [print(i) for i in pops]
    return pops

if __name__ == "__main__":
    decorate(task=1, task_name="stack")
```

Описание работы программы:

Импорты и Настройка:

Скрипт импортирует необходимые модули, включая sys, os и утилитные функции из Lab4.utils.

Он изменяет системный путь, чтобы включить родительский каталог, что позволяет импортировать модули оттуда.

Класс Stack (Стек):

Класс Stack инициализируется с чтением входного файла и сохраняет операции в self.array.

У него есть методы push, pop и result:

push(item): Добавляет элемент в стек.

pop(): Удаляет и возвращает последний элемент из стека.

result(): Проходит по массиву операций, выполняя операции добавления и удаления на основе команд.

Основная Функция:

Функция main() создает экземпляр Stack, обрабатывает операции и записывает вывод.

Она также выводит результаты на консоль.

Точка Входа:

Скрипт проверяет, запускается ли он как основной модуль, и вызывает функцию decorate, которая, используется для логирования или управления задачами..

Тесты:

```
import unittest
from Lab4.task1.src.Stack import *


class TestStack(unittest.TestCase):
    def setUp(self):
        self.stack = Stack()


    def test_case_2(self):
        # given
        self.stack.array = ["3", "+ 10", "+ 1234", "-"]
```

```
        expected_result = ['1234']

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_3(self):
        # given
        self.stack.array = ["3", "+ 5", "+ 10", "-"]
        expected_result = ['10']  # Проверка на правильность извлечения последнего элемента

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_4(self):
        # given
        self.stack.array = ["5", "+ 100", "+ 200", "+ 300", "-", "-"]
        expected_result = ['300', '200']  # Проверка на извлечение двух последних элементов

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_5(self):
        # given
        self.stack.array = ["2", "+ 7", "-"]
        expected_result = ['7']  # Проверка на извлечение единственного элемента

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_6(self):
        #empty input
        # given
        self.stack.array = []
        expected_result = []  # Проверка на извлечение единственного элемента

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

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

Вывод по задаче:

Задача заключалась в создании класса Stack, который реализует основные операции стека: добавление (push) и удаление (pop) элементов. Стек — это структура данных, работающая по принципу "последний пришёл — первый вышел" (LIFO).

Задача №4. Очередь.

Текст задачи:

Реализуйте работу очереди. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо «+ N», либо «–». Команда «+ N» означает добавление в очередь числа N, по модулю не превышающего 10^9. Команда «–» означает изъятие элемента из очереди. Гарантируется, что размер очереди в процессе выполнения команд не превысит 10^6 элементов.

Код программы:

```
import sys
import os
from Lab4.utils import read_input, write_output, decorate

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

class Queue:
    def __init__(self):
        self.queue = []
        self.input_file = read_input(4)
        self.n = int(self.input_file[0])

    def pop(self):
        if len(self.queue) == 0:
            return None
        res = self.queue.pop(0)
        return res

    def push(self, item):
        self.queue.append(item)


    def result(self):
        pops = []
        for i in self.input_file:
            arr = i.split()
            if arr[0] == '+':
                self.push(arr[1])
            elif arr[0] == '-':
                pops.append(str(self.pop()))
        return pops

def main():
    q = Queue()
    res = q.result()
    write_output(4, *res)
    [print(i) for i in res]
```

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

Описание работы программы:

В данном коде реализована структура данных "очередь" (Queue) на языке Python. Очередь работает по принципу "первый пришёл — первый вышел" (FIFO). Код включает в себя основные функции для добавления и удаления элементов из очереди, а также обработку команд из входного файла.

Основные Компоненты:

Импорты:

Код импортирует модули sys, os, а также функции read_input, write_output и decorate из Lab4.utils.

Класс Queue:

Инициализация:

В конструкторе __init__ создается пустой список self.queue для хранения элементов очереди. Также считывается входной файл, и извлекается количество операций n.

Методы:

push(item): Добавляет элемент в конец очереди.

pop(): Удаляет и возвращает элемент из начала очереди. Если очередь пуста, возвращает None.

result(): Обрабатывает команды из входного файла, добавляя элементы с помощью push и удаляя с помощью pop. Результаты удалений собираются в список pops.

Функция main():

Создаёт экземпляр класса Queue, вызывает метод result() для выполнения операций и записывает результаты в выходной файл. Также результаты выводятся на экран.

Точка Входа:

Код проверяет, запускается ли он как основной модуль, и вызывает функцию decorate, которая используется для логирования или управления задачами.

Тесты:

```
import unittest
from Lab4.task4.src.queue import *



class TestQueue(unittest.TestCase):
    def setUp(self):
        self.queue = Queue()


    def test_case_2(self):
```

```
        # given
        self.queue.input_file = ['4', '+ 1', '+ 40', '+ 90', '-']
        expected_result = ['1']

        # when
        result = self.queue.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_3(self):
        # given
        self.queue.input_file = ['5', '+ 100', '+ 200', '-', '+ 300', '-']
        expected_result = ['100', '200']

        # when
        result = self.queue.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_4(self):
        # given
        self.queue.input_file = ['2', '+ 7', '-']
        expected_result = ['7']

        # when
        result = self.queue.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_5(self):
        # given
        self.queue.input_file = ['3', '+ 20', '+ 30', '-']
        expected_result = ['20']

        # when
        result = self.queue.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_6(self):
        # given
        self.queue.input_file = ['3', '+ 10', '-', '-']
        expected_result = ['10', 'None']

        # when
        result = self.queue.result()

        # then
        self.assertEqual(result, expected_result)

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

Вывод по задаче:

Реализация очереди в данном коде является хорошей основой, которая может быть доработана для повышения надежности и удобства использования.

Задача №6. Сортировка пугалом

Текст задачи:

Последовательность A, состоящую из символов из множества «(», «)», «[» и «]», назовем ***правильной скобочной последовательностью***, если выполняется одно из следующих утверждений:

- A – пустая последовательность;
- первый символ последовательности A – это «(», и в этой последовательности существует такой символ «)», что последовательность можно представить как $A = (B)C$, где B и C – правильные скобочные последовательности;
- первый символ последовательности A – это «[», и в этой последовательности существует такой символ «]», что последовательность можно представить как $A = (B)C$, где B и C – правильные скобочные последовательности.

Так, например, последовательности «(())» и «()[]» являются правильными скобочными последовательностями, а последовательности «[)» и «((» таковыми не являются.

Входной файл содержит несколько строк, каждая из которых содержит последовательность символов «(», «)», «[» и «]». Для каждой из этих строк выясните, является ли она правильной скобочной последовательностью.

Код программы:

```
import sys
import os
from Lab4.utils import read_input, write_output, decorate

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

class Stack:
    def __init__(self):
        self.stack = []
        self.input_file = read_input(6)

    def pop(self):
        if len(self.stack) == 0:
            return None
        return self.stack.pop()

    def push(self, item):
        self.stack.append(item)

    def right_pos(self, s):
        stc = Stack()
        if s[0] == ']' or s[0] == ')' or s.count(')') != s.count('(') or s.count(']') != s.count('['):
            return "NO"
        for x in s:
            if x == '(' or x == '[':
```

```
            stc.push(x)
        else:
            nr = stc.pop()
            if (x == ']' and nr == '(') or (x == ')' and nr == '['):
                return "NO"
    return "YES"

  def result(self):
    answers = []
    for i in self.input_file[1:]:
      answers.append(self.right_pos(i))
    return answers

def main():
  stc = Stack()
  res = stc.result()
  write_output(6, *res)
  [print(i) for i in res]

if __name__ == '__main__':
  decorate(task=6, task_name='bracket_sequence_1')
```

Описание работы программы:

В данном коде реализована структура данных "стек" (Stack) на языке Python, которая используется для проверки правильности последовательности скобок. Код включает в себя основные функции для добавления и удаления элементов из стека, а также для анализа строк на наличие корректных пар скобок.

Основные Компоненты:

Импорты:

Код импортирует модули sys, os, а также функции read_input, write_output и decorate из Lab4.utils.

Класс Stack:

Инициализация:

В конструкторе __init__ создаётся пустой список self.stack для хранения элементов стека. Также считывается входной файл, содержащий строки для проверки.

Методы:

pop(): Удаляет и возвращает верхний элемент стека. Если стек пуст, возвращает None.

push(item): Добавляет элемент на верх стека.

right_pos(s): Проверяет корректность последовательности скобок в строке s. Возвращает "YES", если последовательность корректна, и "NO" в противном случае. Включает проверку на наличие несоответствующих пар скобок и их правильный порядок.

Метод result():

Обрабатывает все строки из входного файла (начиная со второго элемента) и собирает результаты проверки в список answers.

Функция main():

Создаёт экземпляр класса Stack, вызывает метод result() для выполнения проверок и записывает результаты в выходной файл. Также результаты выводятся на экран.

Точка Входа:

Код проверяет, запускается ли он как основной модуль, и вызывает функцию decorate, которая может использоваться для логирования или управления задачами.

Тесты:

```
import unittest
from Lab4.task6.src.bracket_sequence_1 import *

class TestStack(unittest.TestCase):
    def setUp(self):
        self.stack = Stack()


    def test_case_1(self):
        # given
        self.stack.input_file = ['1', '[]']
        expected_result = ['YES'] # Пример ожидаемой высоты дерева

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_2(self):
        # given
        self.stack.input_file = ['10', '[]', '[()]', '[(])', '(()', '())', '[]()', '[[]]', '((()))', '(([[]]))', '([)]']
        expected_result = ['YES', 'YES', 'NO', 'NO', 'NO', 'YES', 'YES', 'YES', 'YES', 'NO']  # Ожидаемые результаты
для каждой строки

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_3(self):
        # given
        self.stack.input_file = ['10', '[]', '[()]', '[(])', '(()', '())', '[]()', '[[]]', '((()))', '(([[]]))', '([)]']
        expected_result = ['YES', 'YES', 'NO', 'NO', 'NO', 'YES', 'YES', 'YES', 'YES', 'NO']  # Ожидаемые результаты
для каждой строки

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)
```

```
    def test_case_4(self):
        # given
        self.stack.input_file = ['5', '(())))', '(()())', '(()', '()((', '[[[]]]']
        expected_result = ['YES', 'YES', 'NO', 'NO', 'YES']

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_5(self):
        # given
        self.stack.input_file = ['6', '([{}])', '[()]', '{[()]}', '((({}))', '([]{})', '([)]']
        expected_result = ['YES', 'NO', 'YES', 'NO', 'YES', 'NO']  # Ожидаемые результаты для каждой строки

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_6(self):
        # given
        self.stack.input_file = ['8', '[]()', '([])', '[()]', '(([]))', '((()))', '([', '(()))', '([)]']
        expected_result = ['YES', 'YES', 'YES', 'YES', 'YES', 'NO', 'NO', 'NO']  # Ожидаемые результаты для каждой
строки

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

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

Вывод по задаче:

Реализация стека для проверки последовательности скобок в данном коде является хорошей основой, которая может быть доработана для повышения надежности и удобства использования.

Задача №9. Стек с максимумом

Текст задачи:

Стек - это абстрактный тип данных, поддерживающий операции `Push()` и `Pop()`. Нетрудно реализовать его таким образом, чтобы обе эти операции работали за константное время. В этой задаче ваша цель - реализовать стек, который также поддерживает поиск максимального значения и гарантирует, что все операции по-прежнему работают за константное время.

Реализуйте стек, поддерживающий операции `Push()`, `Pop()` и `Max()`.

Код программы:

```
import sys
import os
from Lab4.utils import read_input, write_output, decorate

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

class Stack:
    def __init__(self):
        self.input_file = read_input(9)
        self.n = int(self.input_file[0])
        self.stack = []
        self.max = None
        self.stack_max = []

    def pop(self):
        """Если в стеки не осталось элементов, то максимумом берется последний элемент из
stack_max"""
        if len(self.stack) == 0:
            self.max = None
            return None
        removed = self.stack.pop()
        if removed == self.max:
            self.stack_max.pop()
            self.max = self.stack_max[-1]
        return removed

    def push(self, item):
        """Если мы добавили в стек 1-й элемент, он становится максимумом, если нет, то новый
элемент сравнивается с текущим максимумом"""
        self.stack.append(item)
        if len(self.stack) == 1 or item > self.max:
            self.max = item
            self.stack_max.append(item)

    def result(self):
        res = []
        for i in self.input_file[1:]:
            d = i.split()
            if d[0] == 'push':
                self.push(int(d[1]))
            elif d[0] == 'pop':
                self.pop()
            elif d[0] == 'max':
                res.append(str(self.max))
        return res

def main():
    stc = Stack()
    res = stc.result()
    write_output(9, *res)
    [print(i) for i in res]

if __name__ == "__main__":
    decorate(task=9, task_name="stack_with_max")
```

Описание работы программы:

В данном коде реализована структура данных "стек" (Stack) с дополнительной функциональностью, позволяющей отслеживать

максимальный элемент в стеке. Это достигается с помощью вспомогательного стека, который хранит максимальные значения на каждом уровне стека. Код включает в себя основные функции для добавления, удаления элементов и получения максимального значения.

Основные Компоненты:

Импорты:

Код импортирует модули sys, os, а также функции read_input, write_output и decorate из Lab4.utils.

Класс Stack:

Инициализация:

В конструкторе __init__ считывается входной файл, содержащий команды для работы со стеком. Также инициализируются переменные: self.stack для хранения элементов стека, self.max для отслеживания текущего максимума и self.stack_max для хранения всех максимальных значений.

Методы:

pop(): Удаляет верхний элемент из стека. Если удаляемый элемент является текущим максимумом, обновляется значение self.max на следующий максимум из self.stack_max.

push(item): Добавляет элемент в стек. Если стек пустой или новый элемент больше текущего максимума, обновляется self.max и добавляется новый максимум в self.stack_max.

result(): Обрабатывает команды из входного файла. В зависимости от команды (push, pop, max) выполняются соответствующие действия. Результаты получения максимума собираются в список res.

Функция main():

Создаёт экземпляр класса Stack, вызывает метод result() для выполнения операций и записывает результаты в выходной файл. Также результаты выводятся на экран.

Точка Входа:

```
Код проверяет, запускается ли он как основной модуль, и вызывает функцию
decorate, которая может использоваться для логирования или управления
задачами.
        Тесты:
import unittest
from Lab4.task9.src.stack_with_max import *
from Lab4.utils import read_input


class TestStack(unittest.TestCase):
    def setUp(self):
        self.stack = Stack()


    def test_case_1(self):
        # given
```

```
        self.stack.input_file = ["3", "push 1", "push 7", "pop"]
        expected_result = []

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_2(self):
        # given
        self.stack.input_file = ["5", "push 3", "push 5", "max", "pop", "max"]
        expected_result = ["5", "3"]  # Ожидаемые результаты для max после операций

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_3(self):
        # given
        self.stack.input_file = ["4", "push 1", "push 7", "pop", "max"]
        expected_result = ["1"]

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_4(self):
        # given
        self.stack.input_file = ["6", "push 10", "push 20", "push 5", "max", "pop", "max"]
        expected_result = ["20", "20"]  # Ожидаемые результаты для max после операций

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_5(self):
        # given
        self.stack.input_file = ["5", "push 15", "push 25", "push 10", "pop", "max"]
        expected_result = ["25"]  # Ожидаемое значение максимума после операций

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_6(self):
        # given
        self.stack.input_file = ["4", "push 5", "push 3", "max", "pop"]
        expected_result = ["5"]  # Ожидаемое значение максимума перед удалением

        # when
        result = self.stack.result()
```

```
        # then
        self.assertEqual(expected_result, result)

    def test_case_7(self):
        # given
        self.stack.input_file = ["7", "push 1", "push 2", "push 3", "max", "pop", "max"]
        expected_result = ["3", "2"]  # Ожидаемые результаты для max после операций

        # when
        result = self.stack.result()

        # then
        self.assertEqual(expected_result, result)


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

Вывод по задаче:

Реализация стека с возможностью отслеживания максимального элемента в данном коде является хорошей основой для работы с данными.

Задача №12. Максимум в движущейся последовательности

Текст задачи:

Задан массив из n целых чисел - $a_1, ..., a_n$ и число $m < n$, нужно найти максимум среди последовательности ("окна") $\{a_i, ..., a_{i+m-1}\}$ для каждого значения $1 \leq i \leq n - m + 1$. Простой алгоритм решения этой задачи за $O(nm)$ сканирует каждое "окно"отдельно.

Ваша цель - алгоритм за $O(n)$.

Код программы:

```
import sys
import os
from Lab4.utils import read_input, write_output, decorate

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

class Queue:
    def __init__(self):
        self.input_file = read_input(12)

        # Проверяем, что входные данные корректны
        try:
            self.n = int(self.input_file[0])
            self.arr = list(map(int, self.input_file[1].split()))
            self.m = int(self.input_file[2])
        except ValueError as e:
            print(f"Ошибка при обработке входных данных: {e}")
            return  # Выход из конструктора, если данные некорректны

        self.queue = []
```

```
        self.max_queue = []  # Стек для хранения максимумов

    def pop(self):
        if self.queue:
            removed = self.queue.pop(0)
            if removed == self.max_queue[0]:
                self.max_queue.pop(0)  # Удаляем максимум из max_queue
            return removed
        return None

    def push(self, item):
        self.queue.append(item)
        # Обновляем max_queue
        while self.max_queue and self.max_queue[-1] < item:
            self.max_queue.pop()  # Удаляем все меньшие элементы из max_queue
        self.max_queue.append(item)

    def result(self):
        res = []
        for i in range(self.n):
            if len(self.queue) < self.m:
                self.push(self.arr[i])
            else:
                res.append(str(self.max_queue[0]))  # Записываем текущий максимум
                self.pop()  # Удаляем элемент
                self.push(self.arr[i])  # Добавляем новый элемент
        if len(self.queue) == self.m:  # Записываем максимум для последней группы
            res.append(str(self.max_queue[0]))
        return res

def main():
    q = Queue()
    res = q.result()
    write_output(12, ' '.join(res))
    print(*res)

if __name__ == "__main__":
    decorate(task=12, task_name="max_in_moving_sequence")
```

Тесты:

```
import unittest
from Lab4.task12.src.max_in_moving_sequence import *
from Lab4.utils import read_input


class TestStack(unittest.TestCase):
    def setUp(self):
        self.queue = Queue()

    def test_case_0(self):
        # given
        self.queue.n = 5
        self.queue.arr = [1, 3, 5, 7, 9]
        self.queue.m = 3
        expected_result = ['5', '7', '9']

        # when
        result = self.queue.result()
```

```
    # then
    self.assertEqual(expected_result, result)

def test_case_1(self):
    # given
    self.queue.n = 6
    self.queue.arr = [2, 4, 1, 3, 5, 6]
    self.queue.m = 4
    expected_result = ['4', '5', '6']  # Максимумы для каждой подгруппы

    # when
    result = self.queue.result()

    # then
    self.assertEqual(expected_result, result)

def test_case_2(self):
    # given
    self.queue.n = 5
    self.queue.arr = [10, 20, 30, 40, 50]
    self.queue.m = 2
    expected_result = ['20', '30', '40', '50']  # Максимумы для каждой подгруппы

    # when
    result = self.queue.result()

    # then
    self.assertEqual(expected_result, result)

def test_case_3(self):
    # given
    self.queue.n = 4
    self.queue.arr = [5, 3, 8, 1]
    self.queue.m = 3
    expected_result = ['8', '8']  # Максимумы для каждой подгруппы

    # when
    result = self.queue.result()

    # then
    self.assertEqual(expected_result, result)

def test_case_4(self):
    # given
    self.queue.n = 7
    self.queue.arr = [1, 2, 3, 4, 5, 6, 7]
    self.queue.m = 5
    expected_result = ['5', '6', '7']  # Максимумы для каждой подгруппы

    # when
    result = self.queue.result()

    # then
    self.assertEqual(expected_result, result)

def test_case_5(self):
    # given
    self.queue.n = 6
    self.queue.arr = [7, 3, 5, 8, 1, 2]
    self.queue.m = 4
```

```
        expected_result = ['8', '8', '8']  # Максимумы для каждой подгруппы

        # when
        result = self.queue.result()

        # then
        self.assertEqual(expected_result, result)

    def test_case_6(self):
        # given
        self.queue.n = 5
        self.queue.arr = [1, 3, 5, 7, 9]
        self.queue.m = 3
        expected_result = ['5', '7', '9']

        # when
        result = self.queue.result()

        # then
        self.assertEqual(expected_result, result)

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

Описание работы кода:

В данном коде реализована структура данных "очередь" (Queue) с дополнительной функциональностью для нахождения максимального элемента в движущейся последовательности фиксированной длины. Это достигается с помощью вспомогательной структуры данных, которая хранит максимальные значения. Код включает в себя основные функции для добавления и удаления элементов из очереди, а также для получения текущего максимума.

Основные Компоненты:

Импорты:

Код импортирует модули sys, os, а также функции read_input, write_output и decorate из Lab4.utils.

Класс Queue:

Инициализация:

В конструкторе __init__ происходит считывание входного файла, который содержит команды для работы с очередью. Инициализируются переменные: self.n — количество элементов, self.arr — массив элементов, self.m — размер окна для движущейся последовательности. Также инициализируются self.queue для хранения элементов очереди и self.max_queue для хранения максимумов.

Добавлена обработка ошибок для проверки корректности входных данных с помощью блока try-except.

Методы:

pop(): Удаляет элемент из начала очереди. Если удаляемый элемент является текущим максимумом, он также удаляется из max_queue.
push(item): Добавляет элемент в очередь. При добавлении элемента обновляется max_queue, удаляя из него все меньшие элементы.
result(): Обрабатывает элементы из self.arr, добавляя их в очередь. Если очередь достигает заданного размера m, записывается текущий максимум и удаляется старый элемент. В конце возвращается список максимумов для каждой группы.
Функция main():
Создает экземпляр класса Queue, вызывает метод result() для выполнения операций и записывает результаты в выходной файл. Также результаты выводятся на экран.
Точка Входа:

Код проверяет, запускается ли он как основной модуль, и вызывает функцию decorate, которая может использоваться для логирования или управления задачами.

Вывод по задаче:
Код предоставляет эффективную реализацию очереди с возможностью отслеживания максимального элемента в движущейся последовательности фиксированной длины.

Вывод

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