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 20

Выполнила:
Толстухина К.А.
К3139

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 год

Задачи по варианту

Задача №2. 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°.

Команда «-» означает изъятие элемента из очереди. Гарантируется, что размер очереди в процессе выполнения команд не превысит 106 элементов.

Листинг кода

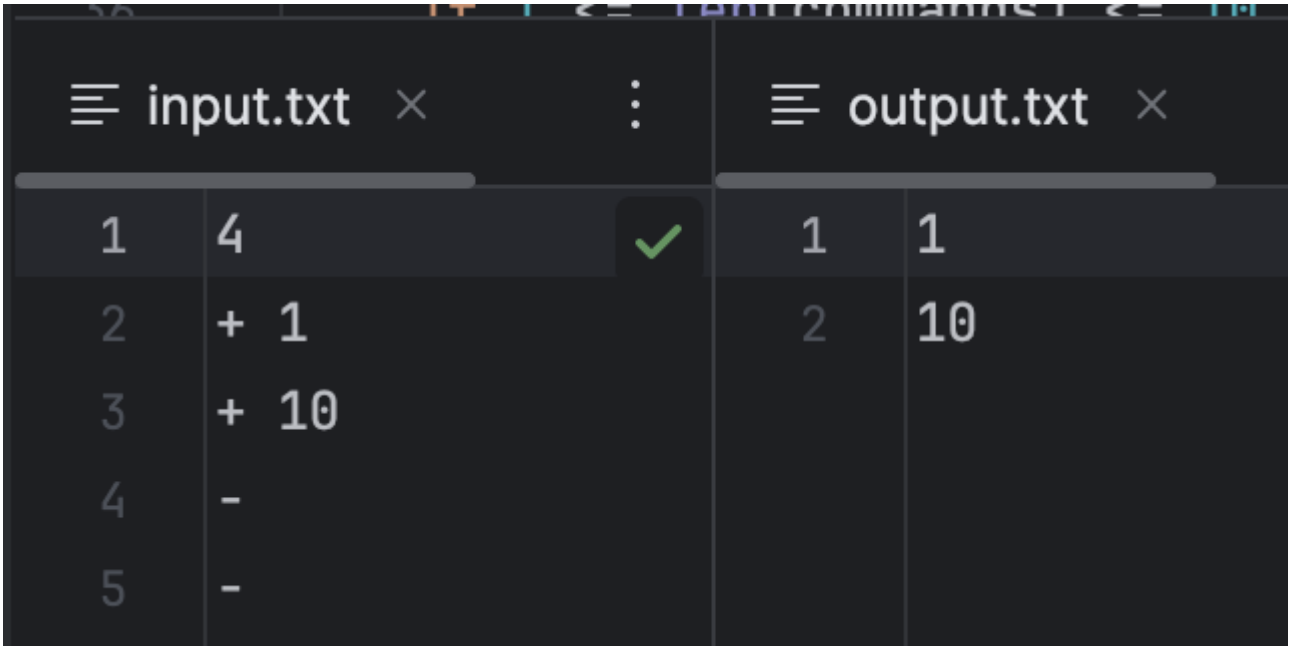
```
import tracemalloc
import time
from lab4.utils import *

t_start = time.perf_counter()
tracemalloc.start()


def algo_queue(commands: list[str]) -> list[str]:
    """
    Обрабатывает команды работы с очередью.
    commands: Список строк, содержащих команды (+ N или -)
    """
    queue = []
    res = []
    for command in commands:
        if command.startswith("+"):
            cmd, number = command.split()
            queue.append(number)
        elif command == "-":
            res.append(queue.pop(0))

    return res
```

```
CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
TXTF_DIR    =    os.path.join(os.path.dirname(CURRENT_DIR),
"txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")


if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    commands = [command.strip() for command in lines[1:]]

    if 1 <= len(commands) <= 10 ** 6:
        results = algo_queue(commands)
        write_file("\n".join(results), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" % (time.perf_counter()
- t_start))
                          print("Затрачено         памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()
```

Текстовое объяснение

Подключаю две библиотеки для отслеживания памяти и времени. Затем Функция для работы с очередью, в нее поступает список с командами, в случае, если команда начинается с “+” то мы добавляем в список queue символ, если мы встречает “-”, то извлекаем первый элемент из очереди и добавляем его в результирующий файл. Далее вне функции я прописываю пути к необходимым директориям/файлам. И в основной части, я считываю данные, проверяю их валидность, вызываю функцию и вывожу время и память.

Пример работы(скрин файлов)



	Время	Память
пример	0.0011385830002836883 секунд	14850 байт

Вывод: был изучен и реализован алгоритм работы очереди

Задача №2. *Скобочная последовательность версия 2*

Текст задачи

Определение правильной скобочной последовательности такое же, как и в задаче 3, но теперь у нас больше набор скобок: [ ]{}().

Нужно написать функцию для проверки наличия ошибок при использовании разных типов скобок в текстовом редакторе типа LaTeX. Для удобства, текстовый редактор должен не только информировать о наличии ошибки в использовании скобок, но также указать точное место в коде (тексте) с ошибочной скобочкой.

В первую очередь объявляется ошибка при наличии первой несовпадающей закрывающей скобки, перед которой отсутствует открывающая скобка, или которая не соответствует открывающей, например, ()[} - здесь ошибка укажет на }.

Во вторую очередь, если описанной выше ошибки не было найдено, нужно указать на первую не совпадающую открывающую скобку, у которой отсутствует закрывающая, например, (в ([].
Если не найдено ни одной из указанных выше ошибок, нужно сообщить, что использование скобок корректно.
Помимо скобок, код может содержать большие и маленькие латинские буквы, цифры и знаки препинания.
Формально, все скобки в коде (тексте) должны быть разделены на пары совпадающих скобок, так что в каждой паре открывающая скобка идет перед закрывающей скобкой, а для любых двух пар скобок одна из них вложена внутри другой, как в (foo [bar]) или они разделены, как в f (a,b) -g [c]. Скобка [соответствует скобке], соответствует и (соответствует) .

Листинг кода

```
import tracemalloc
import time
from lab4.utils import *

t_start = time.perf_counter()
tracemalloc.start()


def checking_bracket_sequence(string: str) -> str:
    """
    обрабатывает входящую строку на наличие правильных
скобочных последовательностей
    :param string: str
    :return: str
    """
    bracket = {")": "(", "}": "{", "]": "["}
    stack = []
    for i, char in enumerate(string, 1):
        if char in "([{":
            stack.append((i, char))
        elif char in "}])":
            if not stack:
```

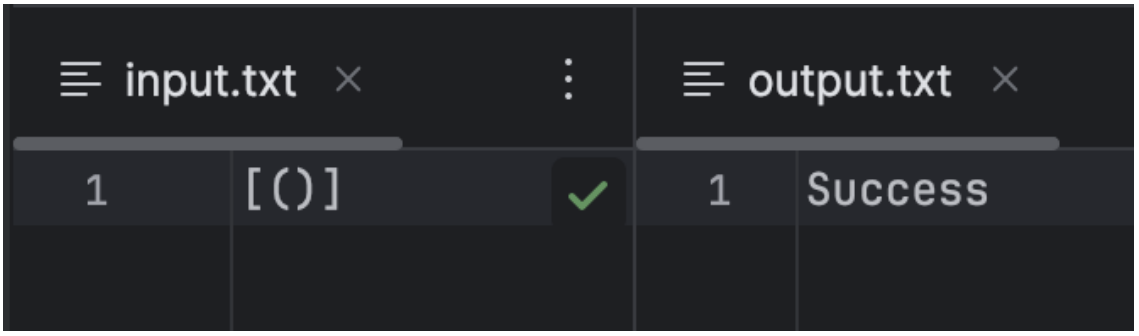
```
                return str(i)
            pos, top_char_stack = stack.pop()
            if top_char_stack != bracket[char]:
                return str(pos)
    if stack:
        pos, top_char_stack = stack.pop()
        return str(pos)
    return "Success"


CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
TXTF_DIR = os.path.join(os.path.dirname(CURRENT_DIR),
"txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")


if __name__ == "__main__":
    string_with_bracket = open_file(INPUT_PATH)[0].strip()

    if 1 <= len(string_with_bracket) <= 10 ** 5:
        results =
checking_bracket_sequence(string_with_bracket)
        write_file(results, OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" % (time.perf_counter()
- t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()
```

Текстовое объяснение

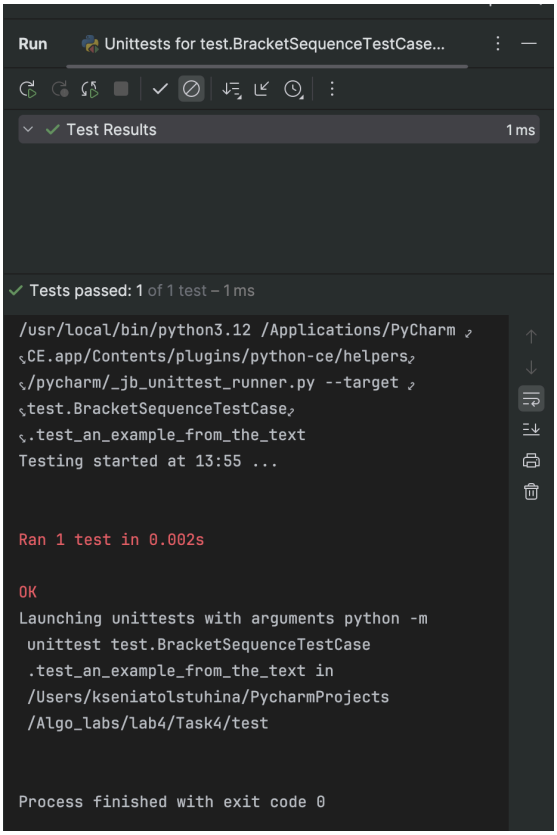
Работа с памятью, временем и файлами аналогична предыдущей задаче. Функция обрабатывает входящую строку, мы проходим по ней, запоминая номер(начиная с 1) и символ, если нам попадается открывающая скобка,

то мы добавляет ее в стек, если закрывающая, мы проверяем стек на заполненность, если в нем нет элементов, значит ошибка, если есть, то мы сравниваем последнюю добавленную скобку, с той, которая должна быть открывающей для текущей.

Пример работы(скрин файлов)




```
def test_an_example_from_the_text(self):
    self.assertEqual(checking_bracket_sequence("[]"), second: "Success")
    self.assertEqual(checking_bracket_sequence("{}[]"), second: "Success")
    self.assertEqual(checking_bracket_sequence("[()]"), second: "Success")
    self.assertEqual(checking_bracket_sequence("(())"), second: "Success")
    self.assertEqual(checking_bracket_sequence("{"), second: "1")
    self.assertEqual(checking_bracket_sequence("{[}"), second: "2")
    self.assertEqual(checking_bracket_sequence("foo(bar);"), second: "Success")
    self.assertEqual(checking_bracket_sequence("foo(bar[i);"), second: "8")
```



	Время	Память
ПРИМЕР	0.0010262089781463146 секунд	14558 байт

Вывод: я поработала с алгоритмом нахождения правильных скобочных последовательностей и применила стек на практике.

Задача №6. Очередь с минимумом

Текст задачи

Реализуйте работу очереди. В дополнение к стандартным операциям очереди, необходимо также отвечать на запрос о минимальном элементе из тех, которые сейчас находится в очереди. Для каждой операции запроса минимального элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда - это либо «+ N», либо «-», либо «?». Команда «+ N» означает добавление в очередь числа N, по модулю не превышающего 10^9.

Команда «-» означает изъятие элемента из очереди. Команда «?» означает запрос на поиск минимального элемента в очереди.

Листинг кода

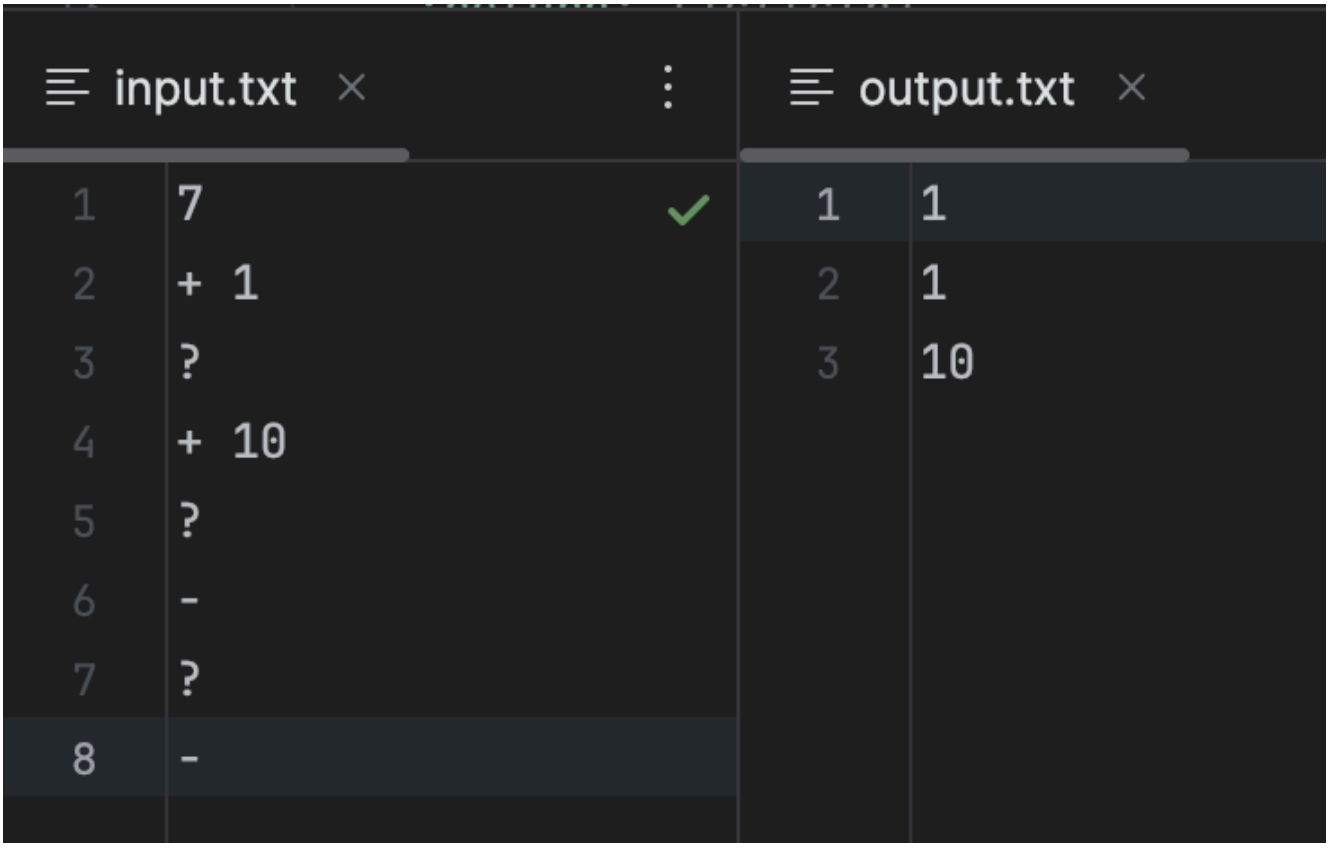
```
import tracemalloc
import time
from lab4.utils import *

t_start = time.perf_counter()
tracemalloc.start()


def algo_queue(commands: list[str]) -> list[str]:
    """
    обрабатывает команды работы с очередью
    :param commands: list[str]
    :return: list[str]
    """
```

```
    queue = []
    res = []
    for command in commands:
        if command.startswith("+"):
            cmd, number = command.split()
            queue.append(number)
        elif command == "-":
            queue.pop(0)
        elif command == '?':
            res.append(str(min(map(int, queue))))
    return res


CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
TXTF_DIR = os.path.join(os.path.dirname(CURRENT_DIR),
"txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")


if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    commands = [command.strip() for command in lines[1:]]

    if 1 <= len(commands) <= 10 ** 6:
        results = algo_queue(commands)
        write_file("\n".join(results), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" % (time.perf_counter()
- t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()
```

Текстовое объяснение

Подключаю две библиотеки для отслеживания памяти и времени. Затем Функция для работы с очередью, в нее поступает список с командами, в случае, если команда начинается с “+” то мы добавляем в список queue символ, если мы встречает “-”, то извлекаем первый элемент из очереди, а если “?” то находим минимум из оставшегося и добавляем его в результирующий файл. Далее вне функции я прописываю пути к необходимым директориям/файлам. И в основной части, я считываю данные, проверяю их валидность, вызываю функцию и вывожу время и память.

Тесты(скрины файлов)



	время	память
тест из условия	0.0008850420126691461 секунд	14979 байт

Вывод: был изучен и реализован дополненный алгоритм работы очереди

Задача №8. Постфиксная запись

Текст задачи

В постфиксной записи (или обратной польской записи) операция записывается после двух операндов. Например, сумма двух чисел A и B записывается как A B +. Запись B C + D * означает привычное нам (B + C) * D, а запись A B C + D
* + означает A + (B + C) * D. Достоинство постфиксной записи в том, что она не требует скобок и дополнительных соглашений о приоритете операторов для своего чтения.
Дано выражение в обратной польской записи. Определите его значение.

Листинг кода

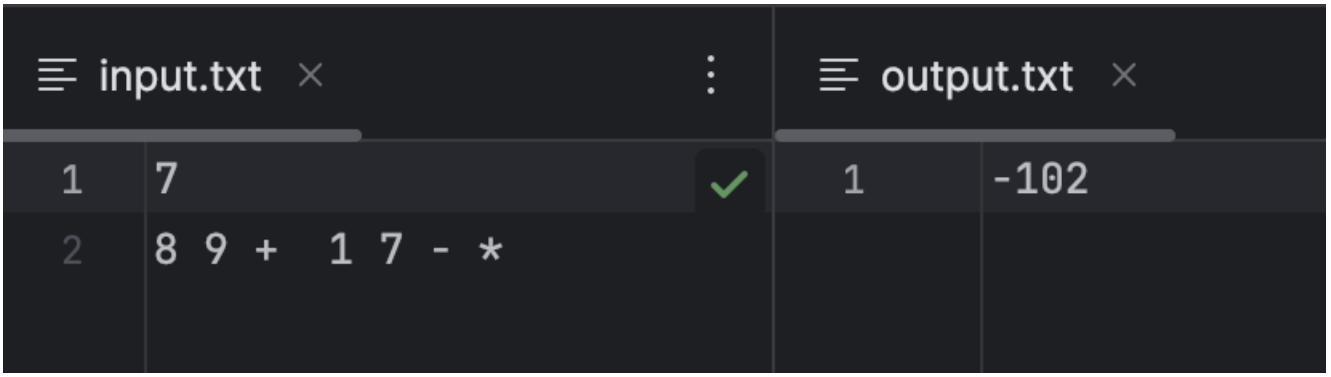
```
import tracemalloc
import time
from lab4.utils import *

t_start = time.perf_counter()
tracemalloc.start()


def postfix_entry(chars: list[str]) -> int:
    """
    вычисляет значение выражения
    :param chars: list[str]
    :return: list[str]
    """
    stack = []
    for char in chars:
        if char in '0123456789':
            stack.append(int(char))
        if char == '+':
            second = stack.pop()
            first = stack.pop()
            stack.append(first + second)
        if char == '-':
            second = stack.pop()
            first = stack.pop()
```

```
            stack.append(first - second)
        if char == '*':
            second = stack.pop()
            first = stack.pop()
            stack.append(first * second)
    return stack[0]


CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
TXTF_DIR = os.path.join(os.path.dirname(CURRENT_DIR),
"txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")


if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    chars = lines[1].split()

    if 1 <= len(chars) <= 10 ** 6:
        results = postfix_entry(chars)
        write_file(str(results), OUTPUT_PATH)
    else:
        print("Введите корректные данные")

    print("Время работы: %s секунд" % (time.perf_counter()
- t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()
```

Текстовое объяснение

Для решения я использую стек. Прохожу по строке, если мне попадается число, то я записываю его в стек. Если попадается арифметический оператор, то я извлекаю из стека два последних числа и их результат записываю снова в конец.

Скрины



	время	память
тест из условия	0.0012220840144436806 секунд	14691 байт

Вывод: я попрактиковала работу со стеком, применяя его на задаче

Дополнительные задачи

Задача №9. Поликлиника

Текст задачи

Очередь в поликлинике работает по сложным правилам. Обычные пациенты при посещении должны вставать в конец очереди. Пациенты, которым "только справку забрать встают ровно в ее середину, причем при нечетной длине очереди они встают сразу за центром. Напишите программу, которая отслеживает порядок пациентов в очереди.

Листинг кода

```
import tracemalloc
import time
from lab4.utils import *

t_start = time.perf_counter()
tracemalloc.start()


def queue_polyclinic(commands: list[str]) -> list[str]:
    """
```

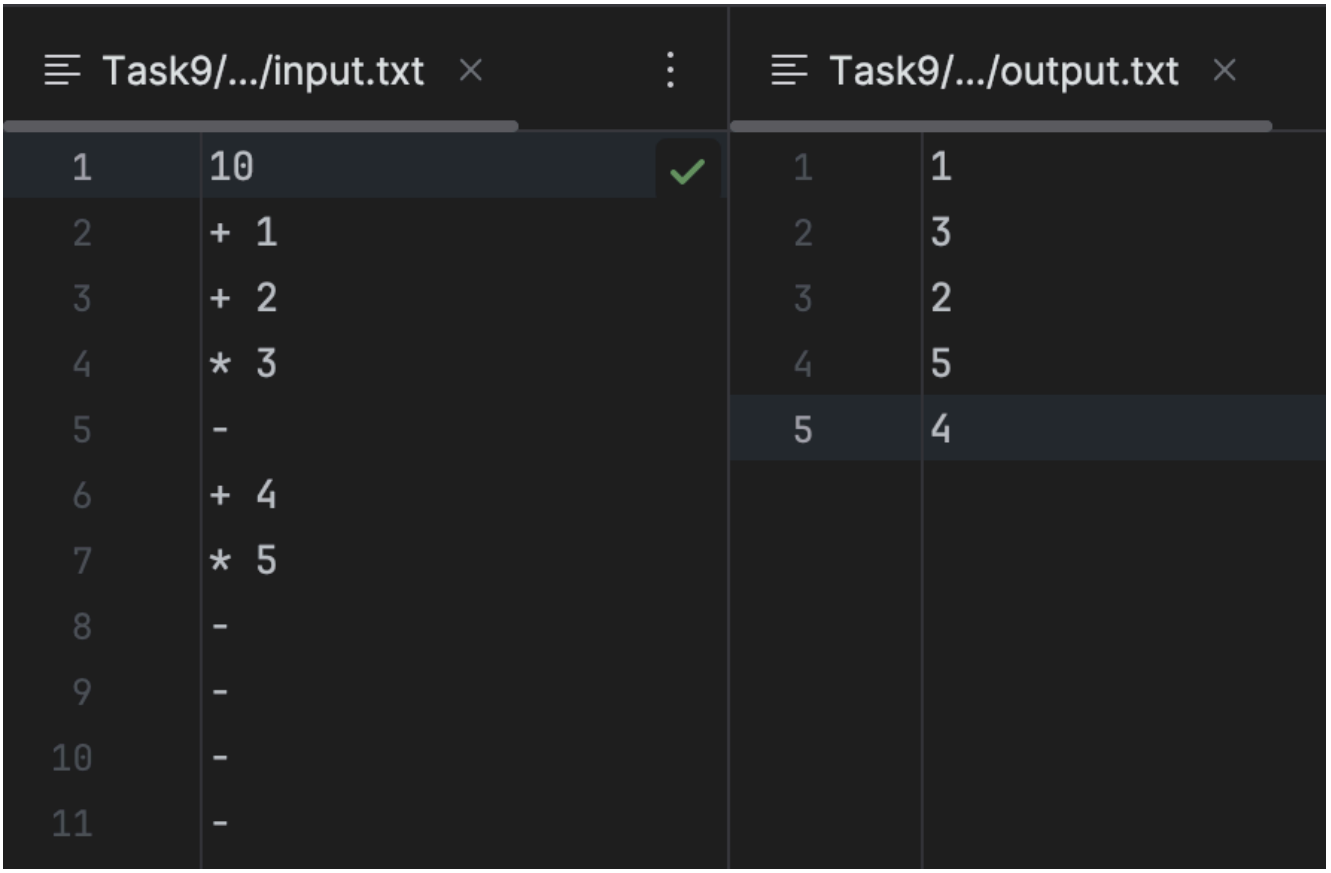
```
    следит за очередью в поликлинике
    :param commands: list[str]
    :return: list[str]
    """
    queue = []
    res = []
    for command in commands:
        if command.startswith("+"):
            queue.append(command.split()[1])
        elif command.startswith("*"):
            if len(queue) % 2 == 0:
                index_mid = len(queue) // 2
            if len(queue) % 2 != 0:
                index_mid = len(queue) // 2 + 1
            queue.insert(index_mid, command.split()[1])
        elif command == '-':
            res.append(queue.pop(0))
    return res


CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
TXTF_DIR = os.path.join(os.path.dirname(CURRENT_DIR),
"txtf")
INPUT_PATH = os.path.join(TXTF_DIR, "input.txt")
OUTPUT_PATH = os.path.join(TXTF_DIR, "output.txt")


if __name__ == "__main__":
    lines = open_file(INPUT_PATH)
    commands = [command.strip() for command in lines[1:]]

    if 1 <= len(commands) <= 10 ** 5:
        results = queue_polyclinic(commands)
        write_file("\n".join(results), OUTPUT_PATH)
    else:
        print("Введите корректные данные")
```

```
    print("Время работы: %s секунд" % (time.perf_counter()
- t_start))
    print("Затрачено памяти:",
tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()
```

Текстовое объяснение

Работая с очередью. в зависимости от входящих данных, либо записываю номера пациентов в конец, либо в середину, сохраняя условие четности/нечетности. В конце возвращаю список тех, кто заходит в кабинет.

Тесты(скрины файлов)



	время	память
тест 1	0.0012220840144436806 секунд	14691 байт

Вывод: хорошая практика для работы с очередью

Задача №13. Реализация стека, очереди

Текст задачи

1) Реализуйте стек на основе связного списка с функциями isEmpty, push, pop и вывода данных.
2) Реализуйте очередь на основе связного списка функциями Enqueue, Dequeue с проверкой на переполнение и опустошения очереди.

Листинг кода

```
class Node:
    """Класс для узла связного списка"""
    def __init__(self, data):
        self.data = data
        self.next = None
```

Объяснение

Связный список — это структура данных, где элементы (узлы) хранятся не подряд, как в массиве, а связаны между собой через указатели. Каждый узел знает только о следующем узле, и для того, чтобы пройти по всему списку, нужно начинать с первого узла (головы) и переходить по цепочке от одного узла к другому. Класс Node подходит для работы как со стеком, так и с очередью, в нем начальные атрибуты, данные узла(элемента) и указатель на следующий

Листинг кода

```
from lab4.Task13.src.ClassNode import Node


class Stack:
    """Класс для стека"""
    def __init__(self):
        self.top = None

    def isEmpty(self) -> bool:
        """Проверка на опустошенность"""
        return self.top is None
```

```
    def push(self, data) -> None:
        """Добавление элемента"""
        new_node = Node(data)
        new_node.next = self.top
        self.top = new_node

    def pop(self):
        """Удаление элемента"""
        if self.isEmpty():
            print("error: stack is empty")
            return
        node_pop = self.top
        self.top = self.top.next
        # node_pop.next = None
        return node_pop.data

    def print_stack(self) -> None:
        """Печать стека"""
        current = self.top
        if self.isEmpty():
            print("stack is empty")
            return
        print("elements:")
        while current:
            print(current.data)
            current = current.next
```

Класс для работы со стеком, взаимодействует с Node, в качестве базовых атрибутов - вершина стека, так как он пока пуст, она - None.
Далее 4 метода: isEmpty(), pop(), push() and print_stack()
Первый проверяет список на наличие элементов, если он пуст, возвращается true, если нет - false.
Второй - удаляет элемент из списка и возвращает его, для начала проверяем, что стек не пуст, далее запоминаем элемент, который надо убрать, сдвигаем узел.
Третий - добавляет элементы, для этого создаем новый узел(элемент), затем связываем новый узел с текущей верхушкой стека. То есть, новый

узел теперь "указан" на тот узел, который был верхушкой стека до этого. Таким образом, новый узел "становится" верхушкой стека, но всё еще сохраняет связь с предыдущими элементами стека. Теперь, когда новый узел связан с предыдущей верхушкой стека, мы обновляем указатель top стека, чтобы он указывал на новый узел. То есть новый узел становится верхушкой стека.
Последний метод печатает наш стек, то есть выводит элементы, пока они есть.

Листинг кода

```
from lab4.Task13.src.ClassNode import Node


class Queue:
    """Класс для очереди"""
    def __init__(self, max_size=None):
        self.front = None
        self.rear = None
        self.current_size = 0
        self.max_size = max_size

    def isEmpty(self) -> bool:
        """Проверка на опустошенность"""
        return self.front is None

    def isFull(self) -> bool:
        """Проверка на полноту"""
        if self.max_size is not None:
            return self.current_size > self.max_size
        return False

    def Enqueue(self, data) -> None:
        """Добавление элемента"""
        if self.isFull():
            print("error: queue is full")
            return
```

```
        new_node = Node(data)
        if self.isEmpty():
            self.front = self.rear = new_node
        else:
            self.rear.next = new_node
            self.rear = new_node

        self.current_size += 1

    def Dequeue(self):
        """Удаление элемента"""
        if self.isEmpty():
            print("error: queue is empty")
            return

        dequeued_data = self.front.data
        self.front = self.front.next

        if self.isEmpty():
            self.rear = None

        self.current_size -= 1
        return dequeued_data

    def print_queue(self):
        """Печать очереди"""
        current = self.front
        if self.isEmpty():
            print("queue is empty")
            return

        print("elements:")
        while current:
            print(current.data)
            current = current.next
```

Теперь аналогичный класс с очередью isEmpty, isFull, print_queue работают точно также, только isFull проверяет не пустоту, а полноту(если она задана)
Enqueue() и Dequeue() работают по такому же принципы, добавляют и удаляют элементы, только Dequeue() убирает элементы(узлы) из начала

Вывод: мной было написано три класса, два из которых реализуют работу стека и очереди и теоретически их в дальнейшем можно использовать для других задач.

Вывод по всей работе: я изучила стек и очередь и научилась применять их на практике.