САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №4
по курсу «Алгоритмы и структуры данных»
Тема: Стек, очередь, связанный список
Вариант 17

Выполнил:
Останин А. С.
К3140

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №1. Стек

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 109 . Команда “–”означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выполнения команд не превысит 106 элементов

```
from lab4.utils import read_data, write_data


def get_stack_deletions(commands: list) -> list:
    """
    Возвращает список удалённых элементов в стеке

    Пример:
    ['10', '-', '3', '1', '-'] -> ['10', '1']
    """
    stack = []
    deleted_elements = []
    for i in commands:
        if i != '-':
            stack.append(i)
        else:
            deleted_elements.append(stack[-1])
            stack = stack[:-1]
    return deleted_elements


def format_input(commands: list) -> list:
    """
    Форматирует входные данные

    Пример:
    ['+ 1', '+ 10', '-', '+2', '+ 1234', '-'] -> ['1', '10', '-', '2',
'1234', '-']
    """
    commands_format = []
    for i in commands:
        i_split = i.split()
        if len(i_split) == 2:
            commands_format.append(i_split[1])
        else:
            commands_format.append(i_split[0])
    return commands_format


def task1():
    path_input = '../txtf/input.txt'
    path_output = '../txtf/output.txt'

    commands = read_data(path_input)[1:]  # не считаем первую строчку, так
```

```
как там количество элементов
    commands_format = format_input(commands)
    deleted_elements = get_stack_deletions(commands_format)
    deleted_elements_str = ', '.join(deleted_elements)  # преобразуем
список в строку

    write_data(path_output, deleted_elements_str)
    print(deleted_elements)


if __name__ == "__main__":
    task1()
```

Здесь мы просто реализуем работу стека, а именно добавляем элемент в начало списка или удаляем первый элемент из него. И по необходимости находим минимальный элемент из стека, используя встроенную функцию min.

```
Input:
6
+ 1
+ 10
-
+ 2
+ 1234
-
Output: 10, 1234
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	5.6599994422867894e-05 C	0.00014495849609375 Mb
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: Реализована работа стека и получения минимального элемента из него.

Задача №3. Скобочная последовательность. Версия 1

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

```python
from lab4.utils import read_data, write_data


def check_brackets(brackets_: str) -> bool:
    """
    Проверяет правильность скобочных последовательностей

    Пример: '()[])' -> False
            '()[()]' -> True
    """
    stack = []
    matching_brackets = {')': '(', ']': '['}
    for i in brackets_:
        if i in '([':
            stack.append(i)  # добавляем в стек скобку, которая ещё не
закрыта
        else:
            # Проверяем в стеке наличие скобки, которая закрывает текущую
            if not stack or stack[-1] != matching_brackets[i]:
                return False
            stack.pop()

    return not stack


def task3():
    PATH_INPUT = '../txtf/input.txt'
    PATH_OUTPUT = '../txtf/output.txt'

    brackets = read_data(PATH_INPUT)
    answer = ''
    for i in brackets[1:]:
        if check_brackets(i):
            answer += 'YES\n'
        else:
            answer += 'NO\n'

    write_data(PATH_OUTPUT, answer)
```

```
    print(answer)


if __name__ == "__main__":
    task3()
```

Проходим по последовательности из скобок, открывающиеся скобки добавляем в стек, а закрывающиеся проверяем на наличие открывающихся того же типа в стеке, и если в стеке есть эта скобка, то удаляем её из стека, а если этой скобки нет, то возвращаем False. Если все скобки совпали, то возвращается True

```
Input: 5
()()
([])
()[
(()]
)(

Output: YES
YES
NO
NO
NO
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	7.34e-05 s	0.0007 mb
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: решена задача на скобочную последовательность

Задача №6. Очередь с минимумом

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

```
from lab4.utils import read_data, write_data


def format_data(commands: list) -> list:
    """
    Форматирует данные под задачу

    Пример:
    ['+ 1', '?', '-', '+ 13'] -> ['1', '?', '-', '13']
    """
    commands_format = []
    for i in commands:
        i_split = i.split()
        if len(i_split) == 2:
            commands_format.append(i_split[1])
        else:
            commands_format.append(i_split[0])
    return commands_format


def get_queue_minimals(commands: list) -> list:
    """
    Получает минимальные элементы из очереди по ходу выполнения команд

    Пример:
    ['1', '?', '-', '13', '4', '?'] -> ['1', '4']
    """
    queue = []
    minimals = []
    for i in commands:
        if i == '?':
            min_element = str(min([int(i) for i in queue]))
            minimals.append(min_element)
        elif i == '-':
            queue.pop(0)
        else:
            queue.append(i)
    return minimals


def task6():
    PATH_INPUT = '../txtf/input.txt'
    PATH_OUTPUT = '../txtf/output.txt'

    commands = read_data(PATH_INPUT)[1:]  # не считаем первую строчку, так как там количество элементов
    commands_format = format_data(commands)

    result = get_queue_minimals(commands_format)
    result_format = '\n'.join(result)

    write_data(PATH_OUTPUT, result_format)
    print(result_format)
```

```
if __name__ == "__main__":
    task6()
```

Реализована работа очереди, а именно в начало списка добавляем элемент, а из конца списка забираем. Также по необходимости получаем минимальный элемент из этого списка

```
Input: 7
+ 1
?
+ 10
?
-
?
-

Output: 1
1
10
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	`0.00024 s`	`0.009 mb`
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: реализована работа очереди, а также нахождения минимального элемента из неё

Задача №11. Бюрократия

В министерстве бюрократии одно окно для приема граждан. Утром в очередь встают n человек, i-й посетитель хочет получить ai справок. За один прием можно получить только одну справку, поэтому если после приема посетителю нужны еще справки, он встает в конец очереди. За время приема министерство успевает выдать m справок. Остальным придется ждать следующего приемного дня. Ваша задача - сказать, сколько еще справок хочет получить каждый из оставшихся в очереди посетитель в тот момент, когда прием закончится. Если все к этому моменту разойдутся, выведите -1.

```
from lab4.utils import read_data, write_data


def get_rest(data, certificates):
    """
    Решает задачу

    Пример: get_rest([1, 2, 3], 5) -> [3, [4, 1, 2]]
    """
    c = 0  # индекс человека в очереди
    for i in range(int(certificates)):
        if not data:  # если закончились люди в очереди
            return -1

        c = c % len(data)  # сбрасываем этот индекс, так как очередь
закончилась
        data[c] = int(data[c]) - 1
        if data[c] == 0:
            data.pop(c)
        c += 1

    return [len(data), data]


def task11():
    PATH_INPUT = '../txtf/input.txt'
    PATH_OUTPUT = '../txtf/output.txt'

    # В первой строке берём второе значение, так как это количество
выдаваемых справок
    certificates_number = read_data(PATH_INPUT)[0].split()[1]
    queue = read_data(PATH_INPUT)[1].split()

    result = get_rest(queue, certificates_number)

    if result == -1:
        write_data(PATH_OUTPUT, str(result))
    else:
        write_data(PATH_OUTPUT, f'{result[0]} \n{str(result[1])[1:-1]}')
    print(result[1])


if __name__ == "__main__":
    task11()
```

Проходим по списку и на каждый проход вычитаем из каждого элемента 1, а если при вычитании получилось 0, то удаляем из списка этот элемент.

```
Input:
4 5
2 5 2 3
Output:
3
4, 1, 2
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	`0.0003 s`	`0.001 mb`
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: решена задача

Вывод

Реализованы алгоритмы работы стека и очереди, а также решены задачи по варианту