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, связный список

Выполнил:
Авдиенко Данила Андреевич
Группа К3140

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Оглавление

Оглавление

Задачи по варианту

Задание № 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.

Код:

```
def stack(n, arr):
    stck = []
    callStck = []
    for i in arr:
        if i[0] == "+":
            stck.append(i[1])
        else:
            callStck.append(stck.pop(-1))
    return callStck


if __name__ == "__main__":
    time_start = time.perf_counter()
    n, arr = read_input_lines("../txtf/input.txt")
    print(n)
    print(arr)
    arr1 = stack(n, arr)
    write_output(arr1, "../txtf/output.txt")
    time_memory_tracking(time_start)
```

Текстовое объяснение решения.

Функция stack получает в поле аргументов n и arr, считанные из файлы, затем создаются два пустых массива stck для добавляемых чисел и callStck для удаляемых чисел, при помощи цикла проходим по каждому элементу arr и выполняем добавление/удаление, опираясь на предоставленную команду

Затраты на работу кода на тестовых данных:

```
Время: 0.00036775
Память:  9.1 МБ

Process finished with exit code 0
```

Для функции написаны модульные тесты при помощи unittest.

Вывод: Реализован алгоритм работы со стеком, который обрабатывает команды добавления и извлечения элементов. Функция протестирована, корректно выполняет операции и сохраняет результаты в файл.

Задание №3. Скобочная последовательность. Версия 1

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:

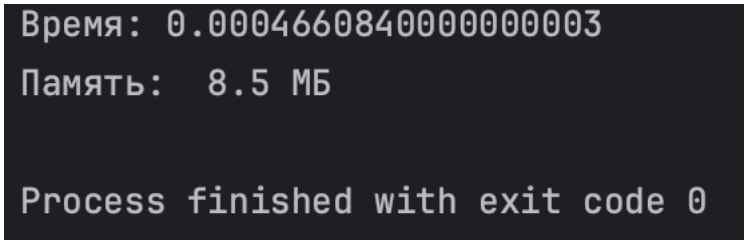
```
def is_valid_parentheses(sequence):
    stack = []
    pairs = {')': '(', ']': '['}
    for char in sequence:
        if char in '([':
            stack.append(char)
        elif char in ')]':
            if not stack or stack[-1] != pairs[char]:
                return "NO"
            stack.pop()
    return "YES" if not stack else "NO"


if __name__ == "__main__":
    time_start = time.perf_counter()
    n, arr = read_input_lines("../txtf/input.txt")
    results = [is_valid_parentheses(''.join(line)) for line in arr]
    write_output(results, "../txtf/output.txt")
    time_memory_tracking(time_start)
```

Текстовое объяснение решения.

Функция is_valid_parentheses принимает в поле аргументов последовательность скобок, внутри создается stack и словарь с парами скобок, пробегая циклом по каждому элементу скобочной последовательности код добавляет в стек открывающие скобки, а при нахождении закрывающих проверяет наличие пары на верхнем месте стека и удаляет в случае нахождения.

Затраты на работу кода на тестовом примере:



Для функции написаны модульные тесты при помощи unittest.

Вывод: Реализован алгоритм проверки корректности скобочной последовательности с использованием стека. Функция успешно обрабатывает входные данные, возвращает правильные результаты (“YES” или “NO”) и сохраняет их в файл.

Задание №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.

Код:

```
def queue_min(n, arr):
    res = []
    queue = []
    for cm in arr:
        if cm[0] == "+":
            queue.append(int(cm[1]))
        elif cm[0] == "-":
            queue.pop(0)
        elif cm[0] == "?":
            mn = 10 ** 10
            for i in queue:
                if i < mn:
                    mn = i
            res.append(mn)
    return res


if __name__ == "__main__":
    time_start = time.perf_counter()
    n, arr = read_input_lines("../txtf/input.txt")
    res = queue_min(n, arr)
    write_output(res, "../txtf/output.txt")
    time_memory_tracking(time_start)
```

Текстовое объяснение решения.

Функция queue_min реализует работу с очередью, поддерживая три команды: добавление элемента (+), удаление первого элемента (-), и нахождение минимального элемента (?). Для команды + число добавляется в конец очереди, для команды - удаляется первый элемент, а для команды ? минимальный элемент ищется перебором текущей очереди. Результаты команд ? добавляются в список и возвращаются.

Затраты на работу кода на тестовом примере:

```
Время: 0.0006905420000000006
Память:  8.6 МБ

Process finished with exit code 0
```

Для функции написаны модульные тесты при помощи unittest.

Вывод: реализован алгоритм работы с очередью, включающий добавление, удаление элементов и нахождение минимального значения. Функция успешно обработала входные данные, корректно выполняет операции и сохраняет результаты в файл.

Задание №7. 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:

```
def max_in_window(n, arr, m):
    dq = []
    res = []
    for i in range(0, n):
        if dq and dq[0] < i - m + 1:
            dq.pop(0)
        while dq and arr[dq[-1]] < arr[i]:
            dq.pop(-1)
        dq.append(i)
        if i >= m-1:
            res.append(arr[dq[0]])
    return res


if __name__ == "__main__":
    time_start = time.perf_counter()
    n, arr, m = read_input("../txtf/input.txt")
    print(n, arr, m)
    arr = max_in_window(n, arr, m)
    write_output(arr, "../txtf/output.txt")
    time_memory_tracking(time_start)
```

Текстовое объяснение решения.

Функция max_in_window ищет максимум в каждом окне длины m в массиве arr длины n. Для этого используется двухсторонняя очередь (dq), которая хранит индексы элементов, обеспечивая быстрый доступ к максимальному элементу текущего окна. На каждой итерации из очереди удаляются устаревшие или меньшие элементы, затем добавляется текущий, а максимальное значение записывается в результат.

Затраты на работу кода на тестовом примере:

```
Время: 0.00045204200000000146
Память:  8.5 МБ

Process finished with exit code 0
```

Для функции написаны модульные тесты при помощи unittest.

Вывод по задаче: Реализован алгоритм нахождения максимума в каждом окне заданной длины с использованием двухсторонней очереди. Функция протестирована и корректно сохраняет результаты в файл.

Вспомогательные файлы. Utils и runall

Utils – содержит в себе переиспользуемые блоки кода для работы с файлами
runall – содержит в себе скрипт для запуска всех тестов лабораторной работы и вывода результата в консоль

Вывод:
Реализованы и протестированы алгоритмы для работы со стеком, очередью, окном и скобками. Все решения протестированы и описаны в md файлах.