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 7

Выполнил:

Дегтярь Г.С.

К3141

Проверила:

Афанасьев А. В

Санкт-Петербург

2024 г.

# Содержание отчета

[**Содержание отчета 2**](#_10udeakjagvs)

[**Задачи по варианту 3**](#_9m6kr4arsrnu)

[Задание 1. Стек. 3](#_5qz11fqbixdy)

[Задание 3. Скобочная последовательность. 5](#_qxk5tpbnbliz)

[Задание 5.Стек с максимумом 8](#_mcd7yyo900u7)

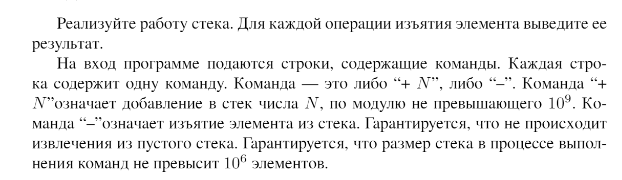
[Задание 7.Максимум в движущейся последовательности 10](#_hc45ccyy60el)

# 

# Задачи по варианту

## Задача №1. Стек.

Текст задачи:



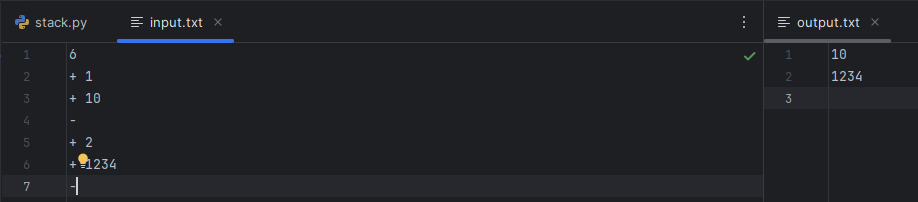


Листинг кода:

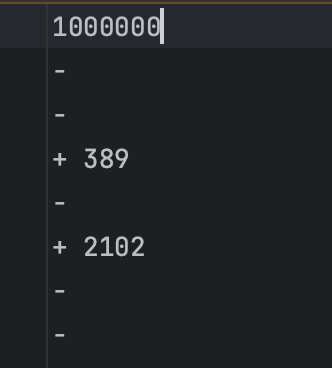
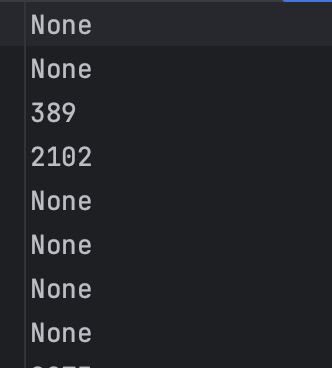
|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  start **=** time**.***perf\_counter***()**  **class** **Stack:**  **def** \_\_init\_\_**(**self**):**  self**.***stack* **=** **[]**  **def** pop**(**self**):**  removed **=** self**.***stack***.***pop***()**  **return** removed  **def** push**(**self**,** item**):**  self**.***stack***.***append***(**item**)**  **with** **open(**"../txtf/input.txt"**,** "r"**)** **as** f**:**  m **=** **int(**f**.***readline***())**  **with** **open(**"../txtf/output.txt"**,** "w"**)** **as** g**:**  stc **=** Stack**()**  **for** i **in** **range(**m**):**  arr **=** f**.***readline***().***split***()**  **if** arr**[**0**]** **==** '+'**:**  stc**.***push***(**arr**[**1**])**  **elif** arr**[**0**]** **==** '-'**:**  g**.***write***(str(**stc**.***pop***())** **+** "\n"**)** |

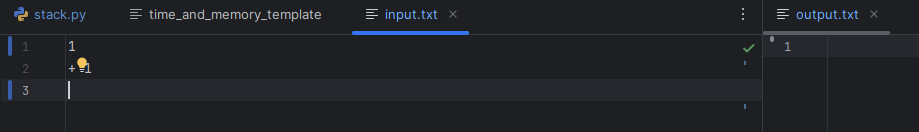
**Текстовое объяснение решения:**

Реализуется класс Stack, поддерживающий функции pop и push. Открывается файл, из него считывается первая строка, содержащая количество операций. Далее пробегается цикл по файлу и если 2 элемент рассматриваемой строки “+”, то применяется push, иначе если “-”, применяется pop.

Результат работы кода на примерах задачи:  


Результат работы кода на максимальных и минимальных значениях:

**** ****



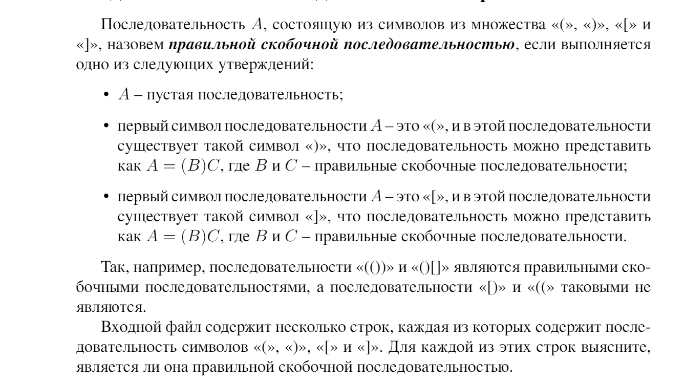
|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0011868999999999977 секунд | 0.04 MB |
| Пример 1 | 0.00027679999999996596 секунд | 0.04 MB |
| Пример 2 |  |  |
| Верхняя граница | 0.329384948938928392 | 0,07 MB |

Вывод о задаче:

Веселая задача, мне понравилась!

**Задача № 3. Скобочная последовательность.**

Текст задачи:



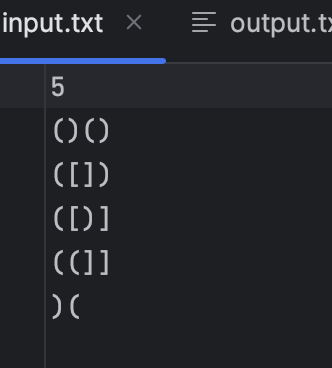
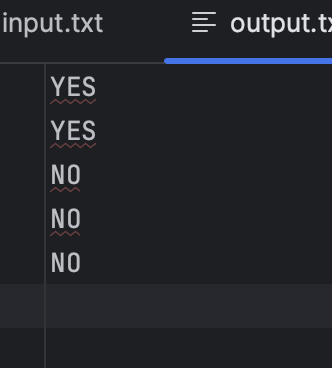
Листинг кода:

|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  start **=** time**.***perf\_counter***()**  **class** **Stack:**  **def** \_\_init\_\_**(**self**):**  self**.***stack* **=** **[]**  **def** pop**(**self**):**  **if** **len(**self**.***stack***)** **==** 0**:**  **return** **None**  removed **=** self**.***stack***.***pop***()**  **return** removed  **def** push**(**self**,** item**):**  self**.***stack***.***append***(**item**)**  **def** right\_pos**(**A**):**  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"  **with** **open(**"../txtf/input.txt"**,** "r"**)** **as** f**:**  n **=** **int(**f**.***readline***())**  **with** **open(**"../txtf/output.txt"**,** "w"**,** encoding**=** 'UTF-8'**)** **as** g**:**  **for** i **in** **range(**n**):**  s **=** f**.***readline***().***strip***()**  g**.***write***(**right\_pos**(**s**)** **+** '\n'**)**  end **=** time**.***perf\_counter***()**  **print(**"Время работы: "**,** end **-** start**,** "секунд"**)**  current**,** peak **=** tracemalloc**.***get\_traced\_memory***()**  **print(**f"Пиковая память: {peak **/** 2**\*\***20:.2f} MB"**)**  tracemalloc**.***stop***()** |

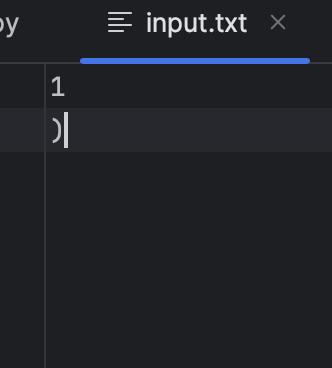
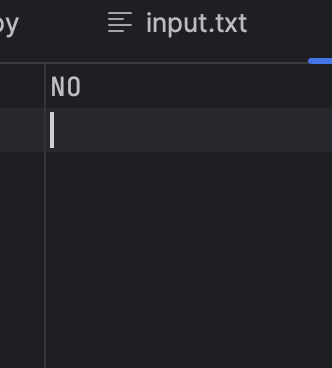
Текстовое объяснение решения:

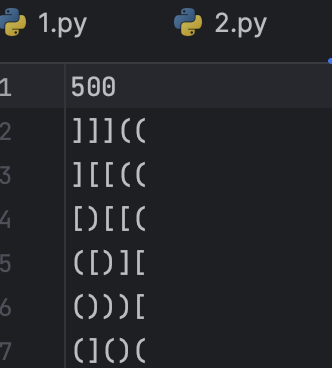
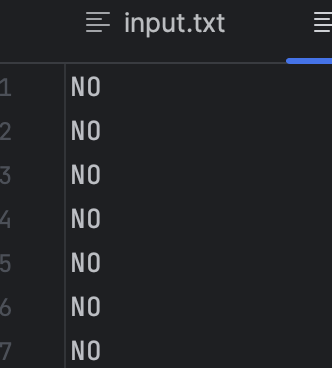
Задача решается при помощи класса stack(). В функции right\_posl перебирается строка со скобочной последовательностью. Если строка начинается с закрывающей скобки, то последовательность считается неправильной. Далее перебираются все скобки в последовательности. Если скобка открывающая, мы добавляем ее в стек. Если же она закрывающая, мы достаем последний элемент из стека. Если он не является соответствующей открывающей скобкой для текущей закрывающей скобки, то последовательность считается неправильной.

Результат работы кода на примерах задачи:

Результат работы кода на максимальных и минимальных значениях:

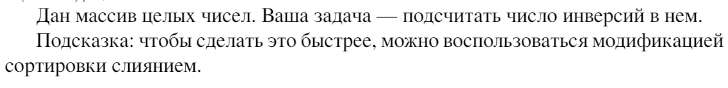
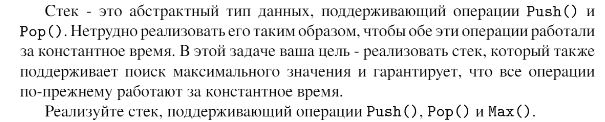
|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0005006999999999096 секунд | 0.04 MB |
| Пример 1 | 0.005204599999999893 секунд | 0.11 MB |
| Пример 2 | 0.0009297000000003663 секунд | 0.05 MB |
| Верхняя граница диапазона значений входных данных из текста задачи | 1.7544864999999996 секунд | 4.19 MB |

Вывод о задаче:

Данная задача показывает, что стек можно применять на практике в самых разных целях.

**Задача № 5. Стек с максимумом.**

Текст задачи:



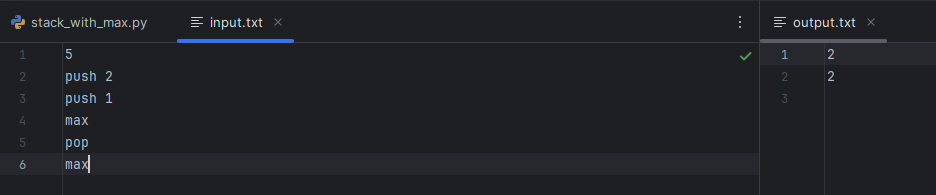
Листинг кода:

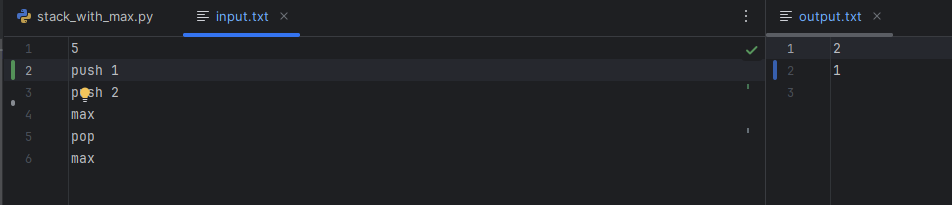
|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  start **=** time**.***perf\_counter***()**  **class** **Stack:**  **def** \_\_init\_\_**(**self**):**  self**.***stack* **=** **[]**  self**.max** **=** **None**  self**.***stack\_max* **=** **[]**  **def** pop**(**self**):**  """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**)**  **with** **open(**"../txtf/input.txt"**,** "r"**)** **as** f**:**  n **=** **int(**f**.***readline***())**  stc **=** Stack**()**  **with** **open(**"../txtf/output.txt"**,** "w"**)** **as** g**:**  **for** i **in** **range(**n**):**  d **=** f**.***readline***().***split***()**  **if** d**[**0**]** **==** 'push'**:**  stc**.***push***(int(**d**[**1**]))**  **elif** d**[**0**]** **==** 'pop'**:**  stc**.***pop***()**  **elif** d**[**0**]** **==** 'max'**:**  g**.***write***(str(**stc**.max)** **+** "\n"**)**  end **=** time**.***perf\_counter***()**  **print(**"Время работы: "**,** end **-** start**,** "секунд"**)**  current**,** peak **=** tracemalloc**.***get\_traced\_memory***()**  **print(**f"Пиковая память: {peak **/** 2**\*\***20:.2f} MB"**)**  tracemalloc**.***stop***()** |

Текстовое объяснение решения:

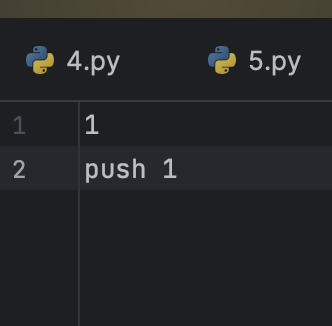
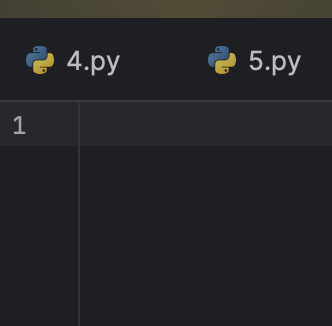
Создается класс stack, поддерживающий функции pop и push. Так же задаются такие свойства стека, как его максимум и список со всеми максимумами. Когда мы добавляем новый элемент в стек, мы проверяем является ли он больше максимума стека. Если является, то максимумом стека становится он + добавляется в список максимумов стека. Когда происходит удаление элемента из стека, если в стеке нет элементов, то максимум становится None. В removed записывается удаляемый из стека элемент( его верхушка ). Если эта верхушка и является максимумом, то она удаляется в том числе из списка максимумов, и новое значение максимума берется как последнее значение в этом списке.

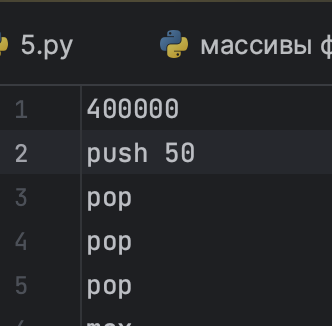
Результат работы кода на примерах из текста задачи:





Результаты работы алгоритма на максимальных и минимальных значениях:

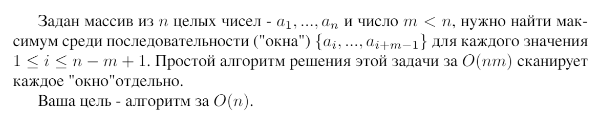
|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00040579999999978966 секунд | 0.04 MB |
| Пример 1 | 0.001330299999999996 секунд | 0.04 MB |
| Пример 2 | 0.0011414000000000007 секунд | 0.04 MB |
| Верхняя граница диапазона значений | 1.9093571999999996 секунд | 15.46 MB |

Вывод о задаче:

Данная задача хорошо демонстрирует принцип работы стека.

**Задача № 7. Макимум в движущейся последовательности.**

Текст задачи:

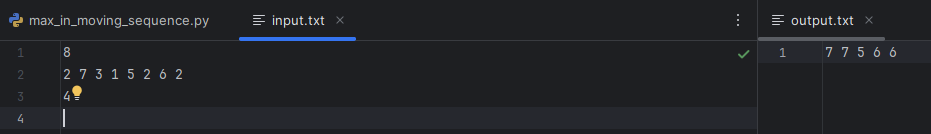


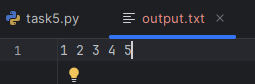
Листинг кода:

|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  start **=** time**.***perf\_counter***()**  **class** **Queue:**  **def** \_\_init\_\_**(**self**):**  self**.***queue* **=** **[]**  self**.***max\_queue* **=** **[]**  **def** pop**(**self**):**  **if** self**.***queue***:**  removed **=** self**.***queue***.***pop***(**0**)**  **if** removed **==** self**.***max\_queue***[**0**]:**  self**.***max\_queue***.***pop***(**0**)**  **return** removed  **return** **None**  **def** push**(**self**,** item**):**  self**.***queue***.***append***(**item**)**  **while** self**.***max\_queue* **and** self**.***max\_queue***[-**1**]** **<** item**:**  self**.***max\_queue***.***pop***()**  self**.***max\_queue***.***append***(**item**)**  **with** **open(**"../txtf/input.txt"**,** "r"**)** **as** f**:**  n **=** **int(**f**.***readline***().***strip***())**  q **=** Queue**()**  **with** **open(**"../txtf/output.txt"**,** "w"**)** **as** g**:**  arr **=** **list(map(int,** f**.***readline***().***split***()))**  m **=** **int(**f**.***readline***())**  **for** i **in** **range(**n**):**  **if** **len(**q**.***queue***)** **<** m**:**  q**.***push***(**arr**[**i**])**  **else:**  g**.***write***(str(**q**.***max\_queue***[**0**])** **+** ' '**)**  q**.***pop***()**  q**.***push***(**arr**[**i**])**  **if** **len(**q**.***queue***)** **==** m**:**  g**.***write***(str(**q**.***max\_queue***[**0**])** **+** ' '**)**  end **=** time**.***perf\_counter***()**  **print(**"Время работы: "**,** end **-** start**,** "секунд"**)**  current**,** peak **=** tracemalloc**.***get\_traced\_memory***()**  **print(**f"Пиковая память: {peak **/** 2**\*\***20:.2f} MB"**)**  tracemalloc**.***stop***()** |

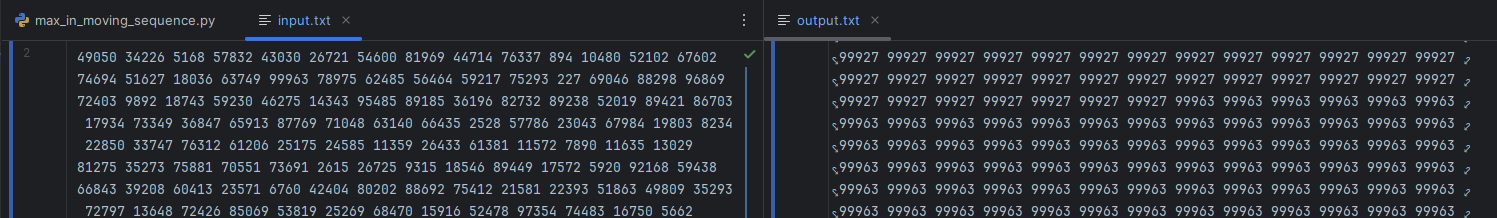
Текстовое объяснение решения:  
Создается класс Queue, который представляет собой очередь с дополнительной структурой данных для хранения максимального элемента в очереди. В методе pop удаляется первый элемент из очереди, а если этот элемент является максимальным, то он также удаляется из структуры данных максимальных элементов. В методе push добавляется новый элемент в очередь, а если он больше максимального элемента в структуре данных, то все элементы, меньшие нового, удаляются из структуры данных, а новый элемент добавляется в конец структуры данных. Далее программа читает из файла input.txt количество элементов n и массив arr, а также количество m. Затем программа создает экземпляр класса Queue и начинает добавлять элементы из массива arr в очередь. Если количество элементов в очереди достигает m, то программа записывает максимальный элемент из очереди в файл output.txt, удаляет первый элемент из очереди и добавляет следующий элемент из массива arr в очередь. После обработки всех элементов из массива arr программа записывает максимальный элемент из очереди в файл output.txt, если количество элементов в очереди равно m. Наконец, программа фиксирует конечное время выполнения и пиковую память, используемую программой, и выводит эти значения на экран.

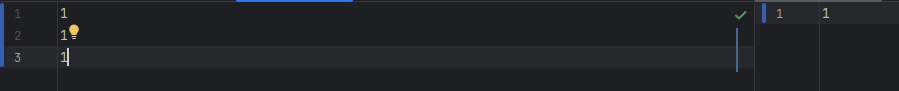
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0016560000000000012 секунд | 0.04 MB |
| Пример 1 | 0. 0.0002775999999995449 | 0.04 MB |
| Пример 2 | 0.0018256000000000001 секунд | 0.04 MB |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.3818564 секунд | 9.38 MB |

Вывод по задаче: Данная задача помогает нам развить навыки работы с очередями.

# Вывод о лабораторной работе:

Данная лабораторная работа направлена на изучение динамических структур: очереди и стека. Данная работа помогла мне освоить навыки работы с подобными алгоритмами.