# 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.

K3141

Проверила:

Афанасьев А. В

Санкт-Петербург 2024 г.

# Содержание отчета

Содержание отчета	
Задачи по варианту	3
Задание 1. Стек	
Задание 3. Скобочная последовательность.	5
Задание 5.Стек с максимумом	8
Задание 7. Максимум в движущейся последовательности	10

# 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.

### Листинг кода:

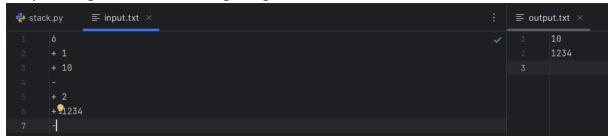
```
import time
import tracemalloc
tracemalloc.start()
start = time.perf counter()
class Stack:
  def __init__(self):
       self.stack = []
  def pop(self):
       removed = self.stack.pop()
       return removed
   def push(self, item):
       self.stack.append(item)
with open("../txtf/input.txt", "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] == '-':
               q.write(str(stc.pop()) + "\n")
```

# Текстовое объяснение решения:

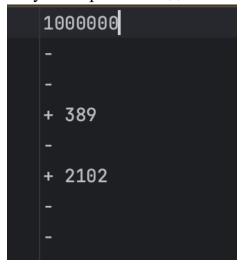
Реализуется класс Stack, поддерживающий функции pop и push. Открывается файл, из него считывается первая строка, содержащая количество операций. Далее пробегается цикл по файлу и если 2

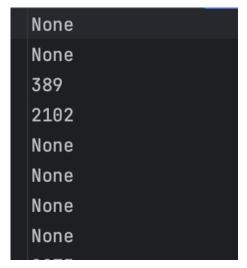
элемент рассматриваемой строки "+", то применяется push, иначе если "-", применяется pop.

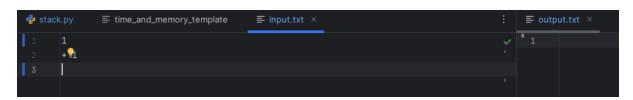
# Результат работы кода на примерах задачи:



# Результат работы кода на максимальных и минимальных значениях:







	Время выполнения	Затраты памяти
Нижняя граница	0.001186899999999977 секунд	0.04 MB
диапазона значений		
входных данных из		
текста задачи		
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:

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

- А 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.

#### 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')

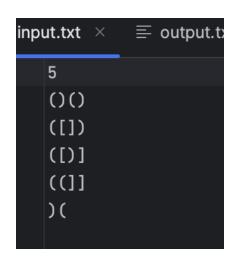
end = time.perf_counter()

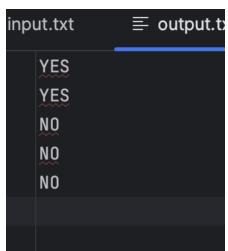
print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} мв")
tracemalloc.stop()
```

# Текстовое объяснение решения:

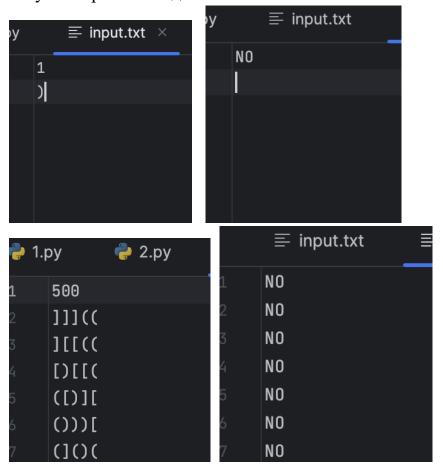
Задача решается при помощи класса stack(). В функции right\_posl перебирается строка со скобочной последовательностью. Если строка начинается с закрывающей скобки, то последовательность считается неправильной. Далее перебираются все скобки в последовательности. Если скобка открывающая, мы добавляем ее 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	0.0005006999999999996 секунд	0.04 MB
диапазона значений		
входных данных из		
текста задачи		
Пример 1	0.005204599999999893 секунд	0.11 MB
Пример 2	0.0009297000000003663 секунд	0.05 MB
Верхняя граница	1.7544864999999996 секунд	4.19 MB
диапазона значений		
входных данных из		
текста задачи		

# Вывод о задаче:

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

# Задача № 5. 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().

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем.

Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

#### 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 q:
        for i in range(n):
          d = f.readline().split()
          if d[0] == 'push':
              stc.push(int(d[1]))
           elif d[0] == 'pop':
```

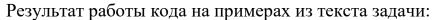
```
stc.pop()
elif d[0] == 'max':
g.write(str(stc.max) + "\n")

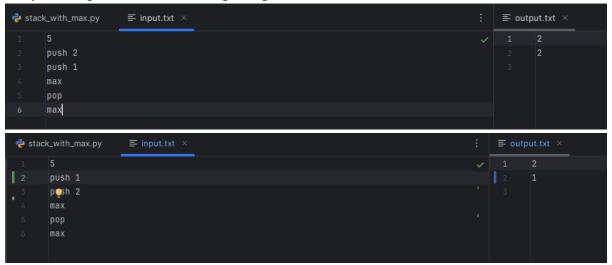
end = time.perf_counter()

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:

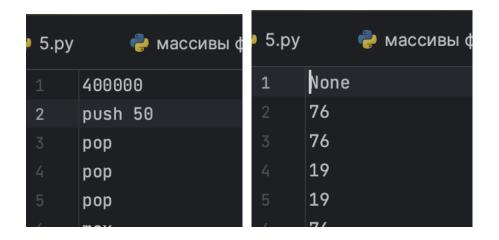
Создается класс stack, поддерживающий функции рор и push. Так же задаются такие свойства стека, как его максимум и список со всеми максимумами. Когда мы добавляем новый элемент в стек, мы проверяем является ли он больше максимума стека. Если является, то максимумом стека становится он + добавляется в 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.





Результаты работы алгоритма на максимальных и минимальных значениях:





	Время выполнения	Затраты памяти
Нижняя граница	0.00040579999999978966 секунд	0.04 MB
диапазона значений		
входных данных из		
текста задачи		
Пример 1	0.00133029999999996 секунд	0.04 MB
Пример 2	0.00114140000000000007 секунд	0.04 MB
Верхняя граница	1.9093571999999996 секунд	15.46 MB
диапазона значений		

#### Вывод о задаче:

Данная задача хорошо демонстрирует принцип работы стека.

# Задача № 7. Макимум 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  $1 \le i \le n-m+1$ . Простой алгоритм решения этой задачи за O(nm) сканирует каждое "окно"отдельно.

Ваша цель - алгоритм за O(n).

#### Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

start = time.perf_counter()

class Queue:
    def __init__(self):
        self.queue = []
        self.max_queue = []
```

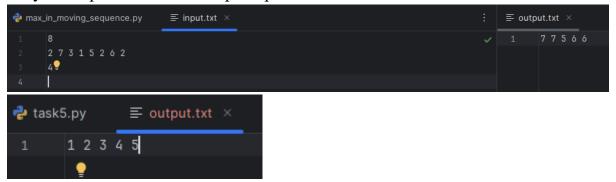
```
def pop(self):
        if self.queue:
            removed = self.queue.pop(0)
            if removed == self.max queue[0]:
                self.max queue.pop(0)
            return removed
        return None
    def push(self, item):
        self.queue.append(item)
        while self.max queue and self.max queue[-1] < item:</pre>
            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:</pre>
                q.push(arr[i])
                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 рор 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 п и массив агт, а также количество т. Затем программа создает экземпляр класса Queue и начинает добавлять элементы из массива агт в очередь. Если количество элементов в очереди достигает т, то программа записывает максимальный элемент из очереди в файл оutput.txt, удаляет первый элемент из очереди и добавляет следующий элемент из массива агт

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	0.0016560000000000012 секунд	0.04 MB
диапазона значений		
входных данных из		
текста задачи		
Пример 1	0. 0.0002775999999995449	0.04 MB
Пример 2	0.0018256000000000001 секунд	0.04 MB
Верхняя граница	0.3818564 секунд	9.38 MB
диапазона значений		
входных данных из		
текста задачи		

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.