

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 21

Выполнила:  
Савченко А.С.

Санкт-Петербург  
2024 г.

## **Содержание отчета**

<b>Содержание отчета</b>	<b>2</b>
<b>Задачи по варианту</b>	<b>3</b>
Задача №1. Стек	3
Задача №3. Скобочная последовательность. Версия 1	5
Задача №5.Стек с максимумом	9
Задача №9. Поликлиника	12
<b>Вывод (по всей лабораторной)</b>	<b>16</b>

## Задачи по варианту

### Задача №1. Стек

Текст задачи:

#### 1 задача. 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  $10^9$ . 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.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится  $M$  ( $1 \leq M \leq 10^6$ ) – число команд. Каждая последующая строка исходного файла содержит ровно одну команду.
- **Формат выходного файла (output.txt).** Выведите числа, которые удаляются из стека с помощью команды “-”, по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из стека. Гарантируется, что изъятий из пустого стека не производится.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
6	10
+ 1	1234
+ 10	
-	
+ 2	
+ 1234	
-	

Листинг кода:

```
def stackk(file):  
    stack = []  
    results = []
```

```

for line in file:
    c = line.strip()
    if c == '-':
        results.append(stack.pop())
    else:
        stack.append(int(c.split()[1]))

return results

if __name__ == "__main__":
    with open('input.txt', 'r') as file:
        m = int(file.readline().strip())
        results = stackk(file)

    with open('output.txt', 'w') as file:
        for result in results:
            file.write(str(result) + '\n')

```

Текстовое объяснение решения:

Создаем пустой стек `stack` для хранения чисел и результирующий в котором будем хранить извлеченные. В цикле в зависимости от считанной команды - или `+` либо удаляем верхний элемент из стека и добавляем его в `results`, либо добавляем элемент в стек

Результат работы кода на примерах из текста задачи:(скрины input output файлов):

input.txt		output.txt
1 6	✓	1 10
2 + 1		2 1234
3 + 10		3
4 -		
5 + 2		
6 + 1234		
7 -		

Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):

The screenshots show the execution of a program on different input ranges. The leftmost image shows the initial state with two input lines and one output line. The middle and right images show a sequence of operations from line 1 to 17, with the rightmost image showing the final result '32'.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0003594 с	14.984375 Мб
Пример из задачи	0.0040779 с	14.8984375 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.3893281 с	16.47265625 Мб

Вывод по задаче:

Результаты тестирования доказывают эффективность структуры данных стек. Алгоритм справляется с обработкой большого объема данных , не превышая заданные ограничения по времени и памяти.

### Задача №3. Скобочная последовательность. Версия 1

Текст задачи:

### 3 задача. 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.

Входной файл содержит несколько строк, каждая из которых содержит последовательность символов «(», «)», «[» и «]». Для каждой из этих строк выясните, является ли она правильной скобочной последовательностью.

- **Формат входного файла (input.txt).** Первая строка входного файла содержит число  $N$  ( $1 \leq N \leq 500$ ) – число скобочных последовательностей, которые необходимо проверить. Каждая из следующих  $N$  строк содержит скобочную последовательность длиной от 1 до  $10^4$  включительно. В каждой из последовательностей присутствуют только скобки указанных выше видов.
- **Формат выходного файла (output.txt).** Для каждой строки входного файла (кроме первой, в которой записано число таких строк) выведите в выходной файл «YES», если соответствующая последовательность является правильной скобочной последовательностью, или «NO», если не является.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
5	YES
()	YES
(())	NO
([])	NO
(([])	NO
)()	

Листинг кода:

```
def brackets(s):  
    stack = []
```

```

for char in s:
    if char in '([':
        stack.append(char)
    elif char == ')':
        if not stack or stack.pop() != '(':
            return "NO"
    elif char == ']':
        if not stack or stack.pop() != '[':
            return "NO"

return "YES" if not stack else "NO"

if __name__ == "__main__":
    with open('input.txt', 'r') as file:
        n = int(file.readline().strip())
        results = []
        for line in file:
            results.append(brackets(line.strip()))

    with open('output.txt', 'w') as file:
        for result in results:
            file.write(result + '\n')

```

Текстовое объяснение решения:

Для каждой строки он поочередно добавляет открывающие скобки в стек и проверяет соответствие закрывающих скобок с вершиной стека. Если находим несовпадение сразу выводим NO. Дойдя до конца проверяем что стек пуст и в этом случае выводим YES

Результат работы кода на примерах из текста задачи:(скрины input output файлов):

input.txt	output.txt
1 5 ✓	1 YES
2 (())	2 YES
3 ([])	3 NO
4 ([)]	4 NO
5 (([])	5 NO
6 )(	6

Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):

input.txt	output.txt
1 1 ✓	1 NO
2 )(	2

input.txt	output.txt
The file size (5 ...	
13 ])]([()([) ✓	184 NO
14 )]]([()([()([	185 NO
15 ]]]([()([()([	186 NO
16 ]]]([()([()([	187 NO
17 ]]]([()([()([	188 NO
18 ]]]([()([()([	189 NO
19 ]]]([()([()([	190 NO
20 ]]]([()([()([	191 NO
21 ]]]([()([()([	192 NO
22 ]]]([()([()([	193 NO
23 ]]]([()([()([	194 NO
24 ]]]([()([()([	195 NO
25 ]]]([()([()([	196 NO
26 ]]]([()([()([	197 NO
27 ]]]([()([()([	198 NO
28 ]]]([()([()([	199 NO

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004895 с	14.7265625 Мб
Пример из задачи	0.0005337 с	14.765625 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.011947 с	15.21875 Мб

Вывод по задаче:



Использование структуры данных стек позволяет эффективно проверять правильность скобочных последовательностей. Он быстро проверяет открывающиеся и закрывающиеся скобки и справляется со всеми тестовыми входными данными.

## 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()`.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится  $n$  ( $1 \leq n \leq 400000$ ) – число команд. Последующие  $n$  строк исходного файла содержит ровно одну команду: `push V`, `pop` или `max`.  $0 \leq V \leq 10^5$ .
- **Формат выходного файла (output.txt).** Для каждого запроса `max` выведите (в отдельной строке) максимальное значение стека.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.
- Пример:

input.txt	output.txt	input.txt	output.txt	input.txt	output.txt
5	2	5	2	3	
push 2	2	push 1	1	push 1	
push 1		push 2		push 7	
max		max		pop	
pop		pop			
max		max			

Листинг кода:

```
class stack_w_max:
    def __init__(self):
        self.stack = []
        self.max_stack = []
```

```

def push(self, value):
    self.stack.append(value)
    if not self.max_stack or value >=
self.max_stack[-1]:
        self.max_stack.append(value)

def pop(self):
    if self.stack:
        value = self.stack.pop()
        if value == self.max_stack[-1]:
            self.max_stack.pop()
        return value

def max(self):
    if self.max_stack:
        return self.max_stack[-1]

if __name__ == "__main__":
    with open('input.txt', 'r') as file:
        n = int(file.readline().strip())
        stack = stack_w_max()
        results = []

        for _ in range(n):
            command = file.readline().strip().split()
            if command[0] == "push":
                stack.push(int(command[1]))
            elif command[0] == "pop":
                stack.pop()
            elif command[0] == "max":
                results.append(stack.max())

    with open('output.txt', 'w') as file:
        for result in results:

```

```
file.write(str(result) + '\n')
```

Текстовое объяснение решения:

Напишем класс "максимального" стека. В нем заводим два списка `stack` со значениями значений, `max_stack` для хранения максимальных значений.

Метод `push`-добавляет значение в стек `stack`, в нем, если значение больше или равно текущему максимуму, оно также добавляется в `max_stack`.

Метод `pop`-удаляет значение из стека `stack`, в нем, удаляемое значение равно текущему максимуму, оно также удаляется из `max_stack`.

Метод `max`-возвращает текущее максимальное значение из стека `max_stack`.

Получение максимального значения из "максимального" стека выполняется за константное время, тк это = возврат последнего элемента из списка

Результат работы кода на примерах из текста задачи:(скрины input output файлов):

input.txt	output.txt	input.txt	output.txt
1 5	1 2	1 5	1 2
2 push 2	2 2	2 push 1	2 1
3 push 1	3	3 push 2	3
4 max		4 max	
5 pop		5 pop	
6 max		6 max	
7			

input.txt	output.txt
1 10	1 9
2 push 2	2 9
3 push 3	3 9
4 push 9	4 9
5 push 7	5
6 push 2	
7 max	
8 max	
9 pop	
10 max	
11 max	
12	

input.txt	output.txt
1 6	1 7
2 push 7	2 7
3 push 1	3
4 push 7	
5 max	
6 pop	
7 max	

Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):

input.txt			output.txt	
1	1	✓	1	None
2	max		2	

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005362 с	14.953125 Мб
Пример из задачи	0.0005048 с	14.8046875 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.1773777 с	15.265625 Мб

Вывод по задаче:

Мне удалось реализовать "максимальный" стек, который также поддерживает поиск максимального значения и гарантирует, что все операции по-прежнему работают за константное время.

Тк все операции выполняются за константное время видно что алгоритм работает очень быстро.

В этой задаче мне понравилось что приведено много тестов для проверки.

## Задача №9. Поликлиника

Текст задачи:

## 9 задача. Поликлиника

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.

- **Формат входного файла (input.txt).** В первой строке записано одно целое число  $n$  ( $1 \leq n \leq 10^5$ ) - число запросов к вашей программе. В следующих  $n$  строках заданы описания запросов в следующем формате:
  - «+ i» – к очереди присоединяется пациент  $i$  ( $1 \leq i \leq N$ ) и встает в ее конец;
  - «\* i» – пациент  $i$  встает в середину очереди ( $1 \leq i \leq N$ );
  - «-» – первый пациент в очереди заходит к врачу. Гарантируется, что на момент каждого такого запроса очередь будет не пуста.

Листинг кода:

```
from time import perf_counter_ns
from psutil import Process
from os import getpid

start_time = perf_counter_ns()

def vvod():
    from collections import deque

    def medecine(commands):
        q = deque()
        results = []
        for c in commands:
            if c.startswith('+'):
                _, i = c.split()
                q.append(i)
            elif c.startswith('*'):
                _, i = c.split()
                mid = (len(q) + 1) // 2
```

```

        q.insert(mid, i)
    elif c.startswith('-'):
        if q:
            results.append(q.popleft())
    return results

if __name__ == "__main__":
    with open('input.txt', 'r') as file:
        n = int(file.readline().strip())
        comm = [file.readline().strip() for _ in range(n)]

    results = medecine(comm)

    with open('output.txt', 'w') as file:
        for result in results:
            file.write(result + '\n')

vvod()
print('Время выполнения:', (perf_counter_ns() - start_time) / 10 ** 9, 'с')
print('Затраты памяти:', Process(getpid()).memory_info().rss / 1024 ** 2, 'МБ')

```

Текстовое объяснение решения:

Создаем двустороннюю очередь в которой будем сохранять пациентов, создаем результирующий список. В зависимости от команды будем выполнять действия: “+” - добавляем пациента в конец очереди .append; “\*” - вычисляем индекс середины очереди, выполняем вставку пациента в середину очереди .insert; “-” удаляем первого пациента .popleft и одновременно возвращаем его номер в результирующий список. Выводим ответ

Результат работы кода на примерах из текста задачи:(скрины input output файлов):

input.txt	output.txt
1 7 ✓	1 1
2 + 1	2 2
3 + 2	3 3
4 -	4
5 + 3	
6 + 4	
7 -	
8 -	

input.txt	output.txt
1 10 ✓	1 1
2 + 1	2 3
3 + 2	3 2
4 * 3	4 5
5 -	5 4
6 + 4	6
7 * 5	
8 -	
9 -	
10 -	
11 -	

Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):

input.txt	output.txt
1 1 ✓	1 1
2 + 1	
3	

input.txt	output.txt
99986 * 91544 ✓	1 33895
99987 -	2 91862
99988 * 14841	3 388
99989 * 95468	4 99982
99990 * 36143	5 29838
99991 -	6 18636
99992 + 51313	7 29777
99993 * 94777	8 84536
99994 -	9 36295
99995 * 35164	10 19003
99996 * 87823	11 62143
99997 * 92270	12 23921
99998 + 39865	13 68470
99999 + 3840	14 40095
100000 -	15 23194
100001 -	16 395
100002	17 80560

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004883 с	14.87890625 Мб

Пример из задачи	0.0008506 с	15.00390625 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.3141589 с	16.328125 Мб

Вывод по задаче:

"Пациенты, которым "только справку забрать встают ровно в ее середину, причем при нечетной длине очереди они встают сразу за центром."  
- ВОТ ИМЕННО ИЗ ЗА ТАКИХ “придумок” возникает беспорядок в больницах!

Ну а вообще оценивая задачу классическая очередь работает достаточно эффективно, но не все ее операции выполняются за константное время.

Чтобы ускорить алгоритм можно использовать встроенную в питон двустороннюю очередь deque. Так время улучшилось с 1.2750351 до 0.3141589 с, на памяти изменения не отразились ,ну это ожидаемо.

### **Вывод (по всей лабораторной)**

Выполняя эту работу я несколько раз использовала такие структуры данных, как стек и очередь, а также их улучшенные версии стек с максимумом и двустороннюю очередь. Я убедилась в эффективности использования этих структур данных при работе с большими входными данными, ведь многие операции с ними имеют линейную сложность.