

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 1

Выполнила:

Гайдук А. С.

К3241

Проверила:

Ромакина О. М.

Санкт-Петербург

2024 г.

## Содержание отчета

Задачи по варианту .....	3
Задача №1. Стек .....	3
Задача №3. Скобочная последовательность. Версия 1 .....	6
Задача №6. Очередь с минимумом.....	9
Задача №7. Максимум в движущейся последовательности.....	12
Дополнительные задачи .....	16
Задача №2. Очередь .....	16
Задача №8. Постфиксная запись .....	18
Задача №13. Реализация стека, очереди и связанных списков .....	21
Вывод .....	24

## 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  $10^9$ . 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()

def stack_operations(commands):
    stack = []
    deleted_elements = []
    for command in commands:
        command = command.strip()
        if command == '-':
            if stack:
                deleted_elements.append(stack.pop())
        elif command.startswith('+'):
            try:
                stack.append(int(command.split()[1]))
            except (IndexError, ValueError):
                print(f"Ошибка: некорректная команда {command}")
                continue
    return deleted_elements

with open('input.txt', 'r') as file:
    lines = file.readlines()
    commands = lines[1:]

start_time = time.perf_counter()

result = stack_operations(commands)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

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

```
print(f"Затраты памяти: {current / 10**6}МВ; Пиковое использование: {peak / 10**6 }МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

### Текстовое объяснение решения:

Для решения этой задачи я создала два массива – удаленные элементы и сам стек. Я удалила лишние пробелы в строках с командами и реализовала логику для добавления и удаления элемента из стека. Для удаления элемента я использовала метод `pop()`, который удаляет и возвращает элемент списка, затем этот элемент с помощью метода `append()` добавляется в список с удаленными элементами. Для реализации добавления элемента в стек я внедрила проверку `startswith`.

### Результат работы кода на примерах из текста задачи:

task1.py	input.txt	output.txt
1	6	
2	+ 1	
3	+ 10	
4	-	
5	+ 2	
6	+ 1234	
7	-	

task1.py	input.txt	output.txt
1	10	
2	1234	

### Результат работы кода на максимальных и минимальных значениях:

task1.py	input.txt	output.txt
1	1	
2	+ 1	

task1.py	input.txt	output.txt
1		

```

task1.py  input.txt  output.txt
⚠ The file size (8 MB) exceeds the configured limit
1 1000000
2 + 999999999
3 -
4 + 999999997
5

```

```

task1.py  input.txt  output.txt
⚠ The file size (5,5 MB) exceeds the configured limit
1 999999999
2 9 9999997
3 999999995
4 999999993

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000016 сек	0.037639 МВ
Пример из задачи	0.000023 сек	0.037822 МВ
Верхняя граница диапазона значений входных данных из текста задачи	1.225265 сек	82.636822 МВ

### Вывод по задаче:

В ходе выполнения этой задачи я ознакомилась с понятием Стек и научилась работать с ним.

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

Последовательность  $A$ , состоящая из символов из множества «(», «)», «[», «]», называется правильной скобочной последовательностью, если выполняется одно из следующих утверждений:

1.  $A$  — пустая последовательность;
2. 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;
3. 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()

OPEN = ('(', '[')
CLOSE = (')', ']')

def check_staples(staples_list):
    result = []
    for string in staples_list:
        is_valid = True
        stack = []
        for s1 in string:
            if s1 in OPEN:
                stack.append(s1)
            elif s1 in CLOSE:
                if not stack:
                    is_valid = False
                    break
                s2 = stack.pop()
                if s2 == "(" and s1 != ")":
                    is_valid = False
                    break
                if s2 == "[" and s1 != "]":
```

```

        is_valid = False
        break
    if stack:
        is_valid = False
        result.append("YES" if is_valid else "NO")
    return result

with open('input.txt', 'r') as f:
    n = int(f.readline())
    staples_list = f.readlines()

start_time = time.perf_counter()

result = check_staples(staples_list)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write("\n".join(str(element) for element in result))

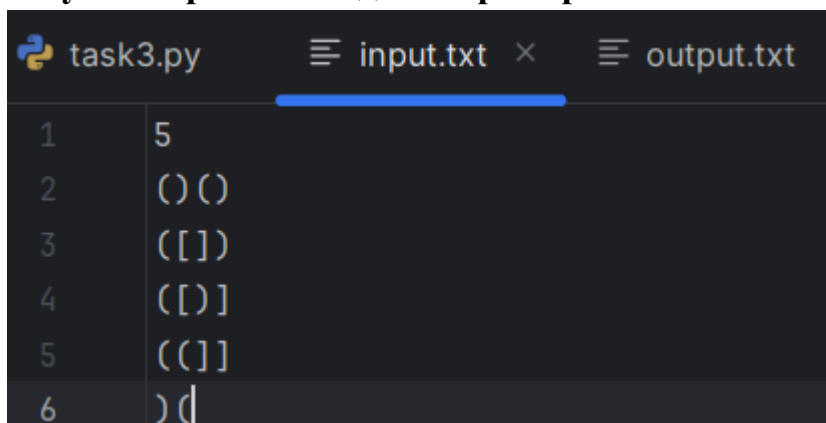
print(f"Затраты памяти: {current / 10**6}MB; Пиковое использование: {peak / 10**6 }MB")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

### Текстовое объяснение решения:

Для решения этой задачи я для каждой строки определила флаг `is_valid` и создала пустой список `stack`. Далее каждый символ строки перебирается, обрабатываются как открывающие, так и закрывающие скобки. Я добавила проверку на соответствие типов скобок.

### Результат работы кода на примерах из текста задачи:



```

task3.py  input.txt  output.txt
1      5
2      ()()
3      ([)]
4      ([)]
5      ([)]
6      )( [

```





текста задачи		
Пример из задачи	0.000020 сек	0.037841 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.849325 сек	5.07487 МВ

### Вывод по задаче:

Я узнала, как реализовать алгоритм для определения правильной скобочной последовательности.

научилась работать с ним.

### 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 находятся 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.

### Листинг кода

```
import time
import tracemalloc
from collections import deque

tracemalloc.start()

def queue_with_min(commands):
    queue = deque()
    min_queue = deque()
    result = []
    for command in commands:
        if command.startswith('+'):
            x = command.split()[1]
            queue.append(x)
            while min_queue and min_queue[-1] > x:
                min_queue.pop()
            min_queue.append(command.split()[1])
```

```

        elif command.startswith('-'):
            if queue:
                removed = queue.popleft()
                if removed == min_queue[0]:
                    min_queue.popleft()

            elif command.startswith('?'):
                if min_queue:
                    result.append(min_queue[0])
                else:
                    result.append(None)
            return result

with open('input.txt', 'r') as f:
    n = int(f.readline())
    commands = f.readlines()

start_time = time.perf_counter()

result = queue_with_min(commands)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write("\n".join(str(element) for element in result))

print(f"Затраты памяти: {current / 10**6}МВ; Пиковое использование: {peak / 10**6 }МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

### Текстовое объяснение решения:

Для решения этой задачи я перебрала каждую команду из списка `commands` с проверкой `startswith`. Для команды «+ N» я реализовала добавление элемента в основную очередь, а также добавила проверку, если новый элемент меньше или равен последнему в `min_queue`, то все элементы, которые больше X, удаляются. Для «-» я реализовала удаление первого элемента из основной очереди и удаление его из `min_queue`, если он там был. Для «?» я реализовала добавление в результат текущего минимального элемента, если он есть. Для эффективной и быстрой работы кода я использовала двустороннюю очередь (`deque`), которая позволяет добавлять и удалять элементы с двух сторон очереди.

### Результат работы кода на примерах из текста задачи:

```
task6.py  input.txt  output.txt
1 7
2 + 1
3 ?
4 + 10
5 ?
6 -
7 ?
8 -|
```

```
task6.py  input.txt  output.txt
1 1
2 1
3 10
```

**Результат работы кода на максимальных и минимальных значениях:**

```
task6.py  input.txt  output.txt
1 1
2 ?
```

```
task6.py  input.txt  output.txt
1 None
```

```
task6.py  input.txt  output.txt
⚠ The file size (7,89 MB) exceeds the configured
1 1000000
2 + 1
3 ?
4 + 2
```

```
task6.py  input.txt  output.txt x
1 1
2 1
3 1
4 1
5 1
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000005 сек	0.037589 МВ
Пример из задачи	0.000018 сек	0.037875 МВ
Верхняя граница диапазона значений входных данных из текста задачи	1.323577 сек	133.72154 МВ

### Вывод по задаче:

Я ознакомилась с такой структурой данных, как очередь, смогла реализовать её работу и внедрила алгоритм поиска минимального значения.

### Задача №7. Максимум в движущейся последовательности

Задан массив из  $n$  целых чисел -  $a_1, \dots, a_n$  и число  $m < n$ , нужно найти максимум среди последовательности ("окна")  $\{a_i, \dots, a_{i+m-1}\}$  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  $O_{(n)}$ .

### Листинг кода

```
import time
import tracemalloc
from collections import deque

tracemalloc.start()
```

```

def sliding_window_maximum(n, arr, m):
    dq = deque()
    result = []
    for i in range(n):
        if dq and dq[0] < i - m + 1:
            dq.popleft()
        while dq and arr[dq[-1]] < arr[i]:
            dq.pop()
        dq.append(i)
        if i >= m - 1:
            result.append(arr[dq[0]])

    return result

with open('input.txt', 'r') as f:
    n = int(f.readline().strip())
    arr = list(map(int, f.readline().strip().split()))
    m = int(f.readline().strip())

start_time = time.perf_counter()

result = sliding_window_maximum(n, arr, m)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write(" ".join(map(str, result)))

print(f"Затраты памяти: {current / 10**6} МБ; Пиковое использование: {peak / 10**6} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

### Текстовое объяснение решения:

Я создала дек для хранения индексов элементов «окна». Проходя по всем элементам массива, я проверяю есть ли в деке индекс, который выходит за пределы текущего окна, если есть, то он удаляется. Далее удаляется индекс, если текущее значение из массива больше, чем значение в массиве, соответствующее индексу в конце дека, тк эти значения больше не могут быть максимумами. После этого текущий индекс добавляется в дек. Когда окно достигает размера  $m$ , я добавляю в результат элемент массива, соответствующий индексу из начала дека, так как он представляет максимум текущего окна. В итоге в массиве результатов хранятся максимумы для всех окон.

### Результат работы кода на примерах из текста задачи:

```
task7.py  input.txt  output.txt
1      8
2      2 7 3 1 5 2 6 2
3      4
```

```
task7.py  input.txt  output.txt
1      7 7 5 6 6
```

**Результат работы кода на максимальных и минимальных значениях:**

```
task7.py  input.txt  output.txt
1      1
2      1
3      1
```

```
task7.py  input.txt  output.txt
1      1
```

```
task7.py  input.txt  output.txt
1      100000
2      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
      29 30 31 32 33 34 35 36 37 38 39
      53 54 55 56 57 58 59 60 61 62 63 64
```

```
task7.py  input.txt  output.txt
1      99999 100000
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000009 сек	0.037664 МВ
Пример из задачи	0.000018 сек	0.03771 МВ

Верхняя граница диапазона значений входных данных из текста задачи	0.119731 сек	9.028426 МВ
---	--------------	-------------

### **Вывод по задаче:**

В этой задаче я использовала дек для хранения индексов, чтобы эффективно находить максимум в каждом окне. Алгоритм работает за  $O(n)$ .

## Дополнительные задачи

### Задача №2. Очередь

Реализуйте работу очереди. 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 очередь числа N, по модулю не превышающего  $10^9$ . Команда «-» означает изъятие элемента из очереди. Гарантируется, что размер очереди в процессе выполнения команд не превысит  $10^6$  элементов.

#### Листинг кода.

```
import time
import tracemalloc
from collections import deque

tracemalloc.start()

def queue(commands):
    q = deque()
    result = []
    for command in commands:
        if command.startswith('+'):
            x = command.split()[1]
            q.append(x)
        elif command.startswith('-'):
            if q:
                result.append(q.popleft())
    return result

with open('input.txt', 'r') as f:
    n = int(f.readline())
    commands = f.readlines()

start_time = time.perf_counter()

result = queue(commands)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write("\n".join(str(element) for element in result))

print(f"Затраты памяти: {current / 10**6}МВ; Пиковое использование: {peak / 10**6 }МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```



### Текстовое объяснение решения:

Для решения этой задачи я преобразовала код из задачи №6, убрав дек `min_queue` и добавляя в массив с результатами удаленные через `popleft()` числа.

### Результат работы кода на примерах из текста задачи:


task2.py	input.txt	output.txt
1	4	
2	+ 1	
3	+ 10	
4	-	
5	-	

task2.py	input.txt	output.txt
1	1	
2	10	

### Результат работы кода на максимальных и минимальных значениях:

task2.py	input.txt	output.txt
1	1	
2	+ 4	

task2.py	input.txt	output.txt
1		

task2.py	input.txt	output.txt
 The file size (8 MB) exceeds the configured limit (2		
1	1000000	
2	+ 999999999	
3	-	
4	+ 999999997	
5	-	

task2.py input.txt output.txt x

⚠ The file size (5,5 MB) exceeds the configured limit (2 MB)

1	999999999
2	999999997
3	999999995
4	999999993

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000011 сек	0.037635 MB
Пример из задачи	0.000008 сек	0.037737 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.522305 сек	85.637487 MB

### Вывод по задаче:

В ходе выполнения данной задачи я закрепила знания по работе с очередью.

### Задача №8. Постфиксная запись

В постфиксной записи (или обратной польской записи) операция записывается после двух операндов. Например, сумма двух чисел  $A$  и  $B$  записывается как  $A B +$ . Запись  $B C + D *$  обозначает привычное нам  $(B + C) * D$ , а запись  $A B C + D * +$  означает  $A + (B + C) * D$ . Достоинство постфиксной записи в том, что она не требует скобок и дополнительных соглашений о приоритете операторов для своего чтения.

Дано выражение в обратной польской записи. Определите его значение.

#### Листинг кода

```
import time
import tracemalloc

tracemalloc.start()

def postfix(equation):
```

```

elements = equation.split()
stack = []
for element in elements:
    if element == '+':
        summ = stack.pop() + stack.pop()
        stack.append(summ)
    elif element == '*':
        product = stack.pop() * stack.pop()
        stack.append(product)
    elif element == '-':
        diff = (stack.pop() - stack.pop()) * -1
        stack.append(diff)
    else:
        stack.append(int(element))
return stack.pop()

with open('input.txt', 'r') as f:
    n = int(f.readline())
    equation = f.readline().strip()

start_time = time.perf_counter()

result = postfix(equation)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write(str(result))

print(f"Затраты памяти: {current / 10**6} МБ; Пиковое использование: {peak / 10**6} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

### Текстовое объяснение решения:

Для решения этой задачи я прописала метод postfix, где числа помещаются в стек, а при встрече оператора извлекаются два верхних элемента. Операция выполняется, и результат возвращается в стек. Так как стек действует по принципу LIFO, при вычитании результат домножается на -1.

### Результат работы кода на примерах из текста задачи:

task8.py	input.txt	output.txt
1	7	
2	8 9 + 1 7 - *	

task8.py	input.txt	output.txt
1	102	

**Результат работы кода на максимальных и минимальных значениях:**

The image displays four sequential screenshots of a code editor window titled 'task8.py'. The editor has tabs for 'input.txt' and 'output.txt'.  
 - The first screenshot shows line 1 with the value '1'.  
 - The second screenshot shows line 1 with the value '1' and a cursor at the end.  
 - The third screenshot shows line 1 with the value '1000000' and line 2 with a long arithmetic expression. A yellow warning banner is visible at the top: 'The file size (5,67 MB) exceeds the configured limit'.  
 - The fourth screenshot shows line 1 with the value '-1'.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000007 сек	0.037456 MB
Пример из задачи	0.000013 сек	0.037456 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.837132 сек	65.837382 MB

### **Вывод по задаче:**

В ходе выполнения этой задачи я узнала, что такое постфиксная запись, закрепила свои знания по работе со стеком.

### Задача №13. Реализация стека, очереди и связанных списков

1. Реализуйте стек на основе связного списка с функциями isEmpty, push, pop и вывода данных.

#### Листинг кода

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

    def __str__(self):
        return f'[{self.data}] -> {self.next}'

class Stack:
    def __init__(self):
        self.head = None

    def push(self, value):
        node = Node(value)
        node.next = self.head
        self.head = node

    def pop(self):
        if self.is_empty():
            return None
        popped = self.head.data
        self.head = self.head.next
        return popped

    def is_empty(self):
        return self.head is None
```

#### Текстовое объяснение решения:

Я создала класс Node, который отвечает за создание элемента, а также вывод результата. В элементе хранятся данные и ссылка на следующий элемент. В классе Stack создается сам стек, а также прописаны методы для добавления и удаления элементов, и их проверки на пустоту.

#### Вывод по задаче:

Я научилась реализовывать стек через связанный список.

2. Реализуйте очередь на основе связного списка функциями Enqueue, Dequeue с проверкой на переполнение и опустошения очереди.

#### Листинг кода

```
class Node:
    def __init__(self, data):
```

```

        self.data = data
        self.next = None

    def __str__(self):
        return f'[{self.data}] -> {self.next}'

class Queue:
    def __init__(self, max_size=None):
        self.head = None
        self.tail = None
        self.max_size = max_size
        self.size = 0

    def is_empty(self):
        return self.head is None

    def is_full(self):
        return self.max_size is not None and self.size >= self.max_size

    def enqueue(self, value):
        if self.is_full():
            print("Переполнение")
            return
        node = Node(value)
        if self.tail is None:
            self.head = self.tail = node
        else:
            self.tail.next = node
            self.tail = node
        self.size += 1

    def dequeue(self):
        if self.is_empty():
            print("Опустошение очереди")
            return None
        dequeued = self.head.data
        self.head = self.head.next
        if self.head is None:
            self.tail = None
        self.size -= 1
        return dequeued

    def __str__(self):
        current = self.head
        result = []
        while current:
            result.append(str(current.data))
            current = current.next
        return " -> ".join(result)

```

### Текстовое объяснение решения:

Я создала класс Node, который отвечает за создание элемента и вывод информации. В элементе хранятся данные и ссылка на следующий элемент. Я создала класс Queue, в котором создается связанный список из элементов.

В этот список можно удалять элементы из начала и добавлять элементы в конец, проверять на пустоту и переполненность.

**Вывод по задаче:**

В этой задаче я научилась реализовывать очередь через связанный список.

## **Вывод**

В ходе выполнения данной лабораторной работы я ознакомилась с такими понятиями, как стек, очередь и связанный список. У меня получилось реализовать различные алгоритмы при работе с этими структурами данных. Сформировалось понимание принципов работы этих структур.