

(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 12

Выполнил:
Колпаков А.С.
К3139

Проверил:
Афанасьев А.В

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №2. Очередь	3
Задача №3. Скобочная последовательность. Версия 2	6
Задача №6. Очередь с минимумом.	9
Задача №12. Строй новобранцев	12
Дополнительные задачи	12
Задача №8. Постфиксная запись	17
Задача №13. Реализация стека, очереди и связанных списков	19
Вывод	25

Задачи по варианту

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 utils

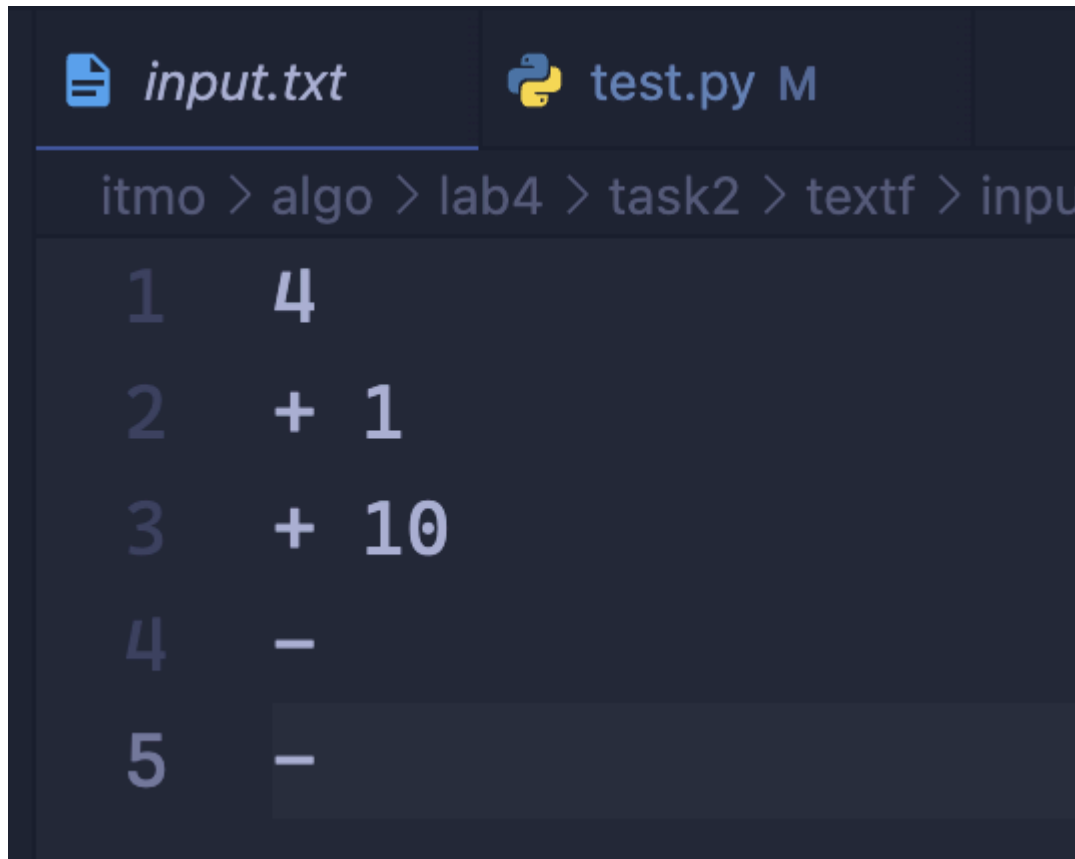
def queue_func(commands):
    queue = []
    front_index = 0
    res = []
    for command in commands:
        if command[0] == "+":
            queue.append(int(command[1]))
        elif command[0] == "-":
            res.append(str(queue[front_index]) + "\n")
            front_index += 1
    return res

if __name__ == '__main__':
    data = utils.read_data('task2/textf/input.txt')
    res = queue_func(data[1:])
    utils.write_file("task2/textf/output.txt", res)
```

Текстовое объяснение решения:

1. Циклом проходимся по командам из заданного списка.
2. Если команда = “+”, то добавляем в очередь значение, если команда ‘-’, то заносим значение текущего начального элемента, после чего делаем начальным элементом следующий элемент.
3. В итоге возвращаем список удаленных элементов.

Результат работы кода:



```
input.txt test.py M
itmo > algo > lab4 > task2 > textf > input.txt
1 4
2 + 1
3 + 10
4 -
5 -
```

 `output.txt`

 `test.py` M

itmo > algo > lab4 > task2 > textf > out

11

210

3|

Тест примера
Время работы: 0.00044837500900030136 секунд
Память: 0.005339622497558594 МБ
.

Ran 1 test in 0.001s
ОК

	Время выполнения	Затраты памяти
Пример из задачи	0.00044837500900030136 секунд	0.005339622497558594 МБ

Вывод по задаче:
Программа успешно реализует работу очереди. Тестирование показало правильную работу алгоритма.

Задача №4. Скобочная последовательность. Версия 2

Определение правильной скобочной последовательности такое же, как и в задаче 3, но теперь у нас больше набор скобок: `[]{}()`.

Нужно написать функцию для проверки наличия ошибок при использовании разных типов скобок в текстовом редакторе типа LaTeX.

Для удобства, текстовый редактор должен не только информировать о наличии ошибки в использовании скобок, но также указать точное место в коде (тексте) с ошибочной скобочкой.

В первую очередь объявляется ошибка при наличии первой несовпадающей закрывающей скобки, перед которой отсутствует открывающая скобка, или которая не соответствует открывающей, например, `()[]` - здесь ошибка укажет на `]`.

Во вторую очередь, если описанной выше ошибки не было найдено, нужно указать на первую несовпадающую открывающую скобку, у которой отсутствует закрывающая, например, `(` в `[]`.

Если не найдено ни одной из указанных выше ошибок, нужно сообщить, что использование скобок корректно.

Помимо скобок, код может содержать большие и маленькие латинские буквы, цифры и знаки препинания.

Формально, все скобки в коде (тексте) должны быть разделены на пары совпадающих скобок, так что в каждой паре открывающая скобка идет перед закрывающей скобкой, а для любых двух пар скобок одна из них вложена внутри другой, как в `(foo[bar])` или они разделены, как в `f(a,b)-g[c]`. Скобка `[` соответствует скобке `]`, соответствует `(` скобке `)`.

Листинг кода:

```
import utils

def check_brackets(data):
    stack = []
    bracket_pairs = {'(': ')', '[': ']', '{': '}'
    for i, char in enumerate(data, start=1):
        if char in "([{":
            stack.append((char, i))
        elif char in ")]}":
            if not stack or stack[-1][0] != bracket_pairs[char]:
                return i
    return i
```

```

        stack.pop()

    if stack:
        return stack[0][1]

    return "Success"

if __name__ == '__main__':
    data = utils.read_data('task4/textf/input.txt')
    res = check_brackets(data[0])
    utils.write_file("task4/textf/output.txt", res)

```

Текстовое объяснение решения:

1. Задаем стэк и пары скобок через словарь.
2. После проходимся по всем значениям заданного слова, если скобка открывающая, то записываем ее индекс и значение в стэк.
3. Если скобка закрывающая, то проверяем, если последний элемент образует пару с этой скобкой. Если это не так, то возвращаем индекс изначальной скобки. В ином случае удаляем этот элемент из стэка.
4. Если в итоге стэк оказался не пуст, то возвращаем индекс оставшегося элемента.
5. Если стэк оказался пустым, то возвращаем 'Success'.

Результат работы кода на примерах из текста задачи:



```
output.txt
itmo > algo > lab4 > t
1 10
```

```
Тест примера
Время работы: 0.0007402080082101747 секунд
Память: 0.0051727294921875 МБ
.
-----
Ran 1 test in 0.001s

OK
```

	Время выполнения	Затраты памяти
Пример из задачи	0.000740208008210174 7 секунд	0.0051727294921875 МБ

Вывод по задаче:

Программа успешно реализует проверку правильной скобочной последовательности и проверку наличия ошибок при использовании разных типов скобок.

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 utils

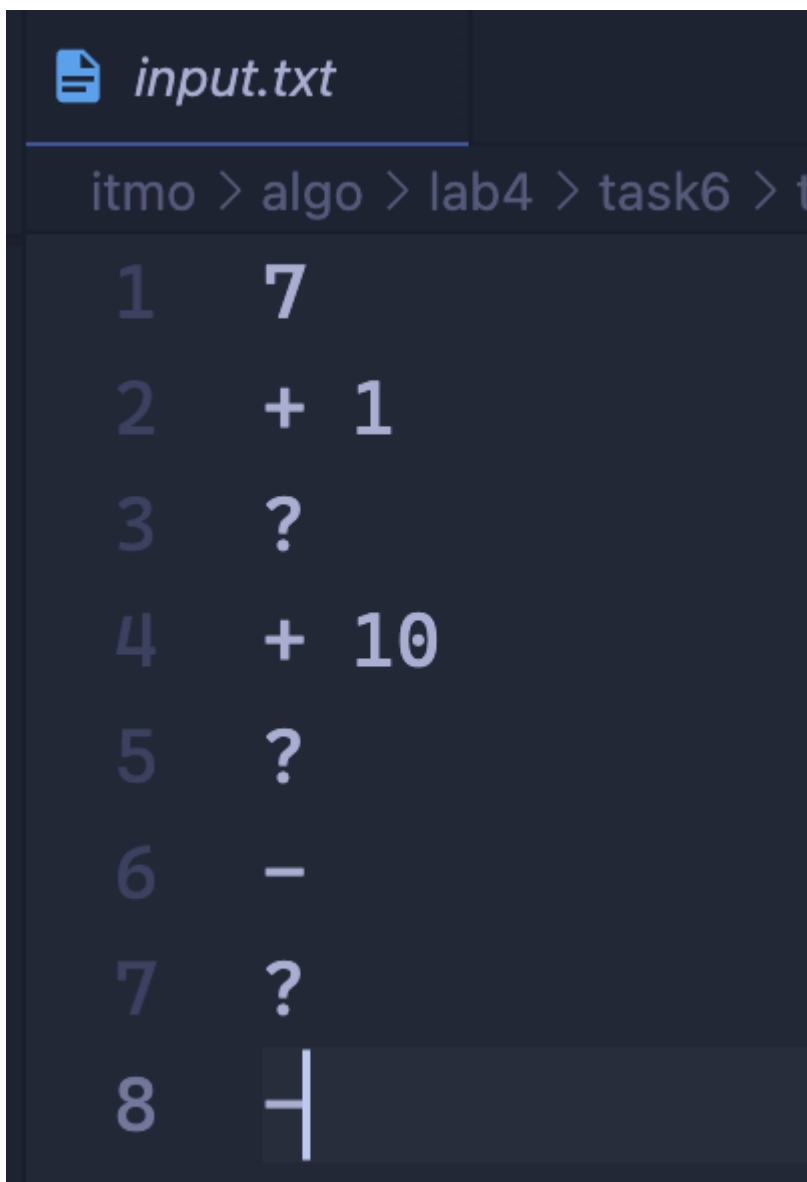
def queue_func_min(commands):
    queue = []
    res = []
    for command in commands:
        if command[0] == "+":
            queue.append(int(command[1]))
        elif command[0] == "-":
            queue.pop(0)
        elif command[0] == "?":
            res.append(str(min(queue)) + "\n")
    return res

if __name__ == '__main__':
    data = utils.read_data('task6/textf/input.txt')
    res = queue_func_min(data[1:])
    utils.write_file("task6/textf/output.txt", res)
```

Текстовое объяснение решения:

1. Действие данного алгоритма во многом схоже с действием алгоритма очереди, но теперь, когда команда = '-', удаляется первый элемент очереди.
2. И при команде = '?', в список результата добавляется минимальное значение текущей очереди.

Результат работы кода на примерах из текста задачи:



```
input.txt
itmo > algo > lab4 > task6 > t
1 7
2 + 1
3 ?
4 + 10
5 ?
6 -
7 ?
8 -
```

```
output.txt
itmo > algo > lab4 > task6

1 1
2 1
3 10
4 |
```

```
Тест примера
Время работы: 0.0003933749976567924 секунд
Память: 0.0053806304931640625 МБ
.
-----
Ran 1 test in 0.001s
OK
```

	Время выполнения	Затраты памяти
Пример из задачи	0.0003933749976567924 секунд	0.0053806304931640625 МБ

Вывод по задаче:

Программа эффективно реализует работу очереди с выводом минимального элемента.

Задача №12. Строй новобранцев

В этой задаче n новобранцев, пронумерованных от 1 до n , разделены на два множества: *строй* и *толпа*. Вначале *строй* состоит из новобранца номер 1, все остальные составляют *толпу*. В любой момент времени строй стоит в один ряд по прямой. Товарищ сержант может использовать четыре команды. Вот они.

- "I, встать в строй слева от J." Эта команда заставляет новобранца номер I, находящегося в толпе, встать слева от новобранца номер J, находящегося в строю.
- "I, встать в строй справа от J." Эта команда действует аналогично предыдущей, за исключением того, что I встает справа от J.
- "I, выйти из строя." Эта команда заставляет выйти из строя новобранца номер I. После этого он присоединяется к толпе.
- "I, назвать соседей." Эта команда заставляет глубоко задуматься новобранца номер I, стоящего в строю, и назвать номера своих соседей по строю, сначала левого, потом правого. Если кто-то из них отсутствует (новобранец находится на краю ряда), то вместо соответствующего номера он должен назвать 0.

Известно, что ни в каком случае строй не остается пустым. Иногда строй становится слишком большим, и товарищ сержант уже не может проверять сам, правильно ли отвечает новобранец. Поэтому он попросил вас написать программу, которая помогает ему в нелегком деле обучения молодежи и выдает правильные ответы для его команд.

Листинг кода:

```
import utils

def recruits_line(n, commands):
    left = {i: 0 for i in range(1, n + 1)}
    right = {i: 0 for i in range(1, n + 1)}

    head = 1
    tail = 1

    results = []

    for command in commands:
```

```
cmd = command[0]

if cmd == "left":
    i, j = int(command[1]), int(command[2])
    left[i] = left[j]
    right[i] = j
    if left[j] != 0:
        right[left[j]] = i
    left[j] = i
    if head == j:
        head = i

elif cmd == "right":
    i, j = int(command[1]), int(command[2])
    right[i] = right[j]
    left[i] = j
    if right[j] != 0:
        left[right[j]] = i
    right[j] = i
    if tail == j:
        tail = i

elif cmd == "leave":
    i = int(command[1])
    if left[i] != 0:
        right[left[i]] = right[i]
    if right[i] != 0:
        left[right[i]] = left[i]
```

```

        if head == i:
            head = right[i]
        if tail == i:
            tail = left[i]
        left[i] = 0
        right[i] = 0

    elif cmd == "name":
        i = int(command[1])
        results.append(f"{left[i]} {right[i]}")

    return results

if __name__ == '__main__':
    data = utils.read_data('task12/textf/input.txt')
    res = recruits_line(data[0][0], data[1:])
    utils.write_file("task12/textf/output.txt", res)

```

Текстовое объяснение решения:

1. Задаем словарь новобранцев слева и справа. Также задаем индекс первого и последнего новобранца = 1.
2. Проходимся по каждой команде из заданных, если команда = 'left', то меняем местами индексы новобранцев в команде, также меняем индекс первого новобранца, если необходимо.
3. Аналогично проделываем и с командой right, меняя индекс новобранца с конца, если необходимо.
4. Если команда = 'leave', то обновляем значения индексов соседних новобранцев, так, чтобы они стали соседними. Также обновляем значения индексов начального и конечного элементов списка.
5. Если команда = 'name', то в список результатов добавляем индексы новобранца слева и справа от текущего.

Результат работы кода на примерах из текста задачи:

```
input.txt
itmo > algo > lab4 > task12 > te
1 3 3
2 left 2 1
3 right 3 1
4 name 1
```

```
output.txt
itmo > algo > lab4 > task1
1 2 3
```

Тест примера

Время работы: 0.0005368749989429489 секунд

Память: 0.005298614501953125 МБ

.

Ran 1 test in 0.001s

OK

	Время выполнения	Затраты памяти
Пример из задачи	0.000536874998942948 9 секунд	0.005298614501953125 МБ

Вывод по задаче:

Алгоритм поиска текущих индексов правого и левого новобранца от текущего эффективно реализует поиск, что подтверждают тесты.

Дополнительные задачи

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 utils

def postfix(expression):
    stack = []

    for token in expression:
        if token.isdigit():
            stack.append(int(token))
        else:
            b = stack.pop()
            a = stack.pop()
            if token == '+':
                stack.append(a + b)
            elif token == '-':
                stack.append(a - b)
            elif token == '*':
                stack.append(a * b)
```

```

    return stack.pop()


if __name__ == '__main__':
    data = utils.read_data('task8/textf/input.txt')
    res = postfix(data[1])
    utils.write_file("task8/textf/output.txt", res)

```

Текстовое объяснение решения:

1. Задаем список стэк.
2. Проходимся по каждому элементу заданного выражения, если текущий элемент является числом, то записываем его в стэк.
3. В ином случае выполняем заданную команду с первыми двумя элементами стэка.
4. В конце у нас остается только результат заданного выражения, которое мы и возвращаем функцией pop.

Результат работы кода на примерах из текста задачи:

 *input.txt*

itmo > algo > lab4 > task8 > textf

1	7
2	8 9 + 1 7 - *

 *output.txt*

itmo > algo > lab4 >

1	-102
---	------

```
Тест примера
Время работы: 0.0009032499947352335 секунд
Память: 0.0052032470703125 МБ
```

```
.
```

```
-----
Ran 1 test in 0.001s
```

```
OK
```

	Время выполнения	Затраты памяти
Пример из задачи	0.000903249994735233 5 секунд	0.0052032470703125 МБ

Вывод по задаче:

Алгоритм эффективно находит значение выражения, заданного в постфиксной форме.

Задача №13. Реализация стека, очереди и связанных списков

1. Реализуйте стек на основе связного списка с функциями isEmpty, push, pop и вывода данных.
2. Реализуйте очередь на основе связного списка функциями Enqueue, Dequeue с проверкой на переполнение и опустошения очереди.
3. Реализуйте односвязный список с функциями вывода содержимого списка, добавления элемента в начало списка, удаления элемента с начала списка, добавления и удаления элемента *после* заданного элемента (key); поиска элемента в списке.
4. Реализуйте двусвязный список с функциями вывода содержимого списка, добавления и удаления элемента с начала списка, добавления и удаления элемента с конца списка, добавления и удаления элемента *до* заданного элемента (key); поиска элемента в списке.

Листинг кода:

stack.py

```
class Node:

    def __init__(self, data):

        self.data = data

        self.next = None

class Stack:

    def __init__(self):

        self.top = None

    def isEmpty(self):

        return self.top is None

    def push(self, data):

        new_node = Node(data)

        new_node.next = self.top

        self.top = new_node

    def pop(self):

        if self.isEmpty():

            return None

        popped_node = self.top

        self.top = self.top.next

        return popped_node.data

    def display(self):

        if self.isEmpty():
```

```

        return

    current = self.top
    while current:
        print(current.data, end=", ")
        current = current.next
    print("None")

if __name__ == "__main__":
    stack = Stack()
    print("Стек пуст:", stack.isEmpty())
    stack.push(10)
    stack.display()
    print("Извлечён элемент:", stack.pop())
    stack.display()
    print("Стек пуст:", stack.isEmpty())

```

Текстовое объяснение решения:

1. Реализация стэка

- a. Задаем узел стэка (Class Node), в котором будет записана непосредственно само значение, а также следующий элемент
- b. Далее задаем класс самого стэка, в котором изначально задаем верхний элемент = None.
- c. Далее задаем функцию isEmpty, которая проверяет, является ли верхний элемент = None, если так, то, соответственно, и сам стэк является пустым.
- d. Также задаем функции push и pop, первая отвечает за добавление нового узла к текущему стэку. А вторая удаляет текущий верхний элемент и возвращает его.
- e. Функция display отображает все элементы стэка.

Результат работы кода на примерах из текста задачи:

Стек пуст: True
10, None
Извлечён элемент: 10
Стек пуст: True

queue.py

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

class Queue:
    def __init__(self, max_size):
        self.first = None
        self.last = None
        self.size = 0
        self.max_size = max_size

    def isEmpty(self):
        return self.size == 0

    def isFull(self):
        return self.size == self.max_size

    def enqueue(self, value):
        if self.isFull():
            return 'Очередь переполнена'
        new_node = Node(value)
        if self.last is None:
```

```

        self.first = self.last = new_node
    else:
        self.last.next = new_node
        self.last = new_node
    self.size += 1

def dequeue(self):
    if self.isEmpty():
        return 'Очередь пуста'
    dequeued_value = self.first.value
    self.first = self.first.next
    if self.first is None:
        self.last = None
    self.size -= 1
    return dequeued_value

def peek(self):
    if self.isEmpty():
        return None
    return self.first.value

def queue_size(self):
    return self.size

if __name__ == "__main__":
    queue = Queue(5)
    queue.enqueue(10)
    print(queue.queue_size())

```

```
print(queue.peek())

queue.dequeue()

print(queue.queue_size())

print(queue.peek())
```

Текстовое объяснение решения:

1. Реализация очереди

- a. Аналогично стэку задаем класс узла.
- b. В классе Queue задаем индекс первого, последнего элементов, также количество элементов и максимальное количество элементов очереди.
- c. Далее задаем функцию проверку очереди на пустоту isEmpty.
- d. Также задаем функцию проверку заполненности очереди isFull.
- e. Также задаем функцию добавления элемента в очередь enqueue, и функцию удаления первого элемента из очереди.
- f. В конце добавляем две функции: функцию вывода первого элемента и вывода количества элементов очереди.

```
1
10
0
None
```

Вывод по задаче:

Данный алгоритм эффективно реализует действие очереди и стэка.

Вывод

В ходе лабораторной работы были изучены основные структуры данных: стек, очередь и связанный список. Были рассмотрены их принципы работы, особенности реализации и области применения. Полученные знания помогут эффективно использовать эти структуры в решении прикладных задач.