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:
Кан Алина
К3139

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

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.

• **Формат входного файла (input.txt).** В первой строке входного файла содержится M ($1 \le M \le 10^6$) – 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 + 1 + 10 - + 2 + 1234 -	10 1234

```
f = open(input.txt)
stack = deque
removed = []
a = []

while True:
    l = f.readline()
    if not l:
        break
    a.append(l.split())

for i in range(len(a)):
    if a[i][0] == '+':
        stack.append[i][1]
    elif a[i][0] == '-':
        removed.append(str(stack[-1]))
        stack.pop()

o = open('output.txt', 'w')
o.write(''.join(str(i) for i in removed))
o.close()
```

Данная программа решает задачу 1. Первое алгоритм открывает файл “input.txt”, создает stack класса “deque” и два пустых массива.
(класс deque() модуля collections возвращает новый объект deque(), инициализированный слева направо данными
из итерируемой последовательности iterable). В первый пустой массив (removed) будем записывать те элементы которые нужно изъять из входного файла, а во второй массив (a) перепишем все команды из входного файла (input.txt).

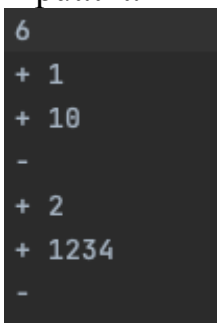
В цикле “while” в пустой массив «a» переписываются команды из входного файла, если же в файле нет команд, программа заканчивает работу.

Далее программа проходит по всем элементам в массиве «a». Если команда содержит “+” в stack добавляем число N, если же команда содержит “-“ в массив removed добавляется предыдущий элемент из массива a, а из массива stack, с помощью команды pop возвращаем значение элемента с индексом i и удаляем его из последовательности.

Затем открываем выходной файл “output.txt” (режим “w” открывает файл на запись, удаляет содержимое файла и, если такого файла не существует, создает новый файл), записываем данные из файла removed и закрываем выходной файл.

	Время выполнения
Пример из задачи	0.0017385650426149368 сек.

Input.txt: Output.txt:



Вывод по задаче:

В этой задаче реализована работа стека с вводом и выводом в файлы. Также алгоритм протестирован на время выполнения алгоритма.

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 106 элементов.

• **Формат входного файла (input.txt).** В первой строке содержится M ($1 \le M \le 10^6$) – число команд. В последующих строках содержатся команды, по одной в каждой строке.

• **Формат выходного файла (output.txt).** Выведите числа, которые удаляются из очереди с помощью команды «–», по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из очереди. Гарантируется, что извлечения из пустой очереди не производится.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

• Пример:

Input.txt	Output.txt
4 + 1 + 10 - -	1 10

```
class node:
    def __init__(self, value = None):
        self.value = value
        self.next = None
    def __str__(self):
        return str(self.value)

class queue:
    def __init__(self):
        self.head = None
        self.tail = None

    def pop(self):
        if self.head:
            host = self.head
            self.head = self.head.next
            return host

    def push(self, value):
        n = Node(value)
        if self.head == None:
            self.head = n
            self.tail = n
            return None
        self.tail.next = n
        self.tail = n

if __name__ == '__main__':
    mas = []
    que = queue()
    with open('input.txt') as f:
        while True:
            l = f.readline()
            if not l:
                break
            mas.append(l.split())

    with open('output.txt', 'w') as o:
        for i in range(5):
            if mas[i][0] == '+':
                que.push(mas[i][1])
            elif mas[i][0] == '-':
                o.write(f'{que.pop()}\n')
```

Данная программа решает задачу 2. Алгоритм считывает данные из входного файла ‘input.txt’, реализует работу очереди и записывает в выходной файл ‘output.txt’ числа, которые удаляются из очереди.

Input.txt:

```
4
+ 1
+ 10
-
-
```

Output.txt:

```
1
10
```

	Время выполнения
Пример из задачи	0.0052032070234417915 сек.

Вывод по задаче:

В данной задаче алгоритм реализует работу очереди и выводит удаленные из очереди числа в выходной файл. Также алгоритм протестирован на время выполнения.

Задача №3 “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 ($1 \le N \le 500$) – 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

```
open_sc = ['[', '(']
close_sc = [']', ')']

f = open('input.txt')
num = []
stack = deque()
remove = []
while True:
    l = f.readline()
    if not l:
        break
    num.append(l.split())

def check(str_):
    stack = []
    for i in str_:
        if i in open_sc:
            stack.append(i)
        elif i in close_sc:
            p = close_sc.index(i)
            if (len(stack) > 0) and (open_sc[p] == stack[len(stack) - 1]):
                stack.pop()
            else:
                return 'NO'
    if len(stack) == 0:
        return 'YES'
    else:
        return 'NO'

o = open('output.txt', 'w')

for i in range(1, len(num)):
    if check(str(num[i])) == 'YES':
        o.write('YES \n')
    if check(str(num[i])) == 'NO':
        o.write('NO \n')
```

Данная программа решает задачу 3. Первые два массива состоят из открытых и закрытых скобок. Создаем stack класса “deque” и два пустых массива. (класс deque() модуля collections возвращает новый объект deque(), инициализированный слева направо данными
из итерируемой последовательности iterable). Также создаем пустой массив num. В него будем записывать скобочные последовательности из входного файла.

В цикле “while” в пустой массив «num» переписываются команды из входного файла, если же в файле нет команд, программа заканчивает работу.

В начале функции “check” создается пустой массив, в который будут записываться скобки из массива num.

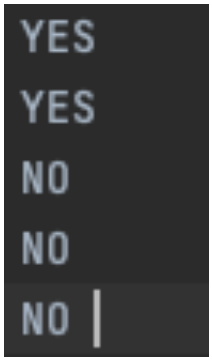
Проходимся по элементам массива, если элемент массива существует в массиве с открытыми скобками, добавляем его в пустой массив stack, далее, если в массиве существует элемент из массива с закрытыми скобками и выполняется условие при котором длина массива “stack” больше нуля и элемент из массива с открытыми скобками с индексом элемента из массива с закрытыми скобками есть в массиве “stack”, с помощью команды pop возвращаем значение элемента с индексом i и удаляем его из последовательности, если условие не выполняется, то возвращается “NO”.

Далее, если длина массива равна нулю возвращается “YES”, если нет, то возвращается “NO”.

После открывается выходной файл “output.txt”. Проверяем все скобочные последовательности через функцию “check”. Если функция возвращает “YES”, в выходной файл записывается “YES”, если функция возвращает “NO”, в выходной файл записывается “NO”.

input.txt:

output.txt:



	Время выполнения
Пример из задачи	0.0016935490421019495 сек.

Вывод по задаче:

В данной задаче алгоритм производит правильную скобочную последовательность с выводом и вводом в файлы. Также алгоритм протестирован на время выполнения.

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 указанный 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 и (соответствует) .

• **Формат входного файла (input.txt).** Входные данные содержат одну строку S, состоящую из больших и маленьких латинских букв, цифр, знаков препинания и скобок из набора [] (). Длина строки S – 1 ≤ S leq10^5.

• **Формат выходного файла (output.txt).** Если в строке S скобки используются правильно, выведите «Success» (без кавычек). В противном случае выведите отсчитываемый от 1 индекс первой несовпадающей закрывающей скобки, а если нет несовпадающих закрывающих скобок, выведите отсчитываемый от 1 индекс первой открывающей скобки, не имеющей закрывающей.

• Ограничение по времени. 5 сек.

• Ограничение по памяти. 256 мб.

• Примеры:

Input.txt	Output.txt
[]	Success
{}[]	Success
[()]	Success
(())	Success
{	1
{[}	3
Foo(bar);	Success
Foo(bar[i);	10

```
f = open('input.txt', 'r')
s = f.readline()
f.close()
stack = []
bracket = {')': '(', '}': '{', ']': '['}

f = open('output.txt', 'w')

for i in range(len(s)):
    el = s[i]
    if el == '(' or el == '[' or el == '{':
        stack += [(el, i)]
    elif el == ')' or el == ']' or el == '}':
        if len(stack) == 0 or bracket[el] != stack[-1][0]:
            f.write(str(i + 1))
            exit(0)
        stack.pop()

if len(stack) != 0:
    f.write(str(stack[0][1] + 1))
else:
    f.write('Success')

f.close()
```

Программа решает задачу 4. Первое, алгоритм считывает данные с файла “input.txt” и закрывает файл “input.txt”. Создаем пустой массив stack,

куда будем записывать элементы входного файла и определять правильность скобочной последовательности. Также создаем массив, в который запишем набор скобок. Открываем выходной файл “output.txt”.

Запускаем цикл “for”. Если i-тый элемент в строке из входного файла одна из открытых скобок, добавляем его в пустой массив stack, если элемент в строке одна из закрытых скобок и выполняется условие при котором длина массива stack равна нулю или элемента в наборе скобок не равен предыдущему в stack’е, в выходной файл записываем индекс, к которому добавляем 1. Затем, с помощью команды pop, возвращаем значение элемента с индексом i и удаляем его из последовательности.

Далее, если длина массива stack’а не равна нулю, в выходной файл указываем на индекс несовпадающей скобки в массиве, а иначе в выходной файл записываем “Success”

Input.txt:	Output.txt:
`[]`	`Success`
`{}[]`	`Success`
`[()]`	`Success`
`(())`	`Success`
`{`	`1`
`{[}`	`3`
`foo(bar);`	`Success`
`foo(bar[i);`	`10`

Примеры из задач	Время выполнения
1	0.000677051953971386 сек.
2	0.0005047879531048238 сек.
3	0.000897857011295855 сек.
4	0.0008141779690049589 сек.
5	0.0016821129829622805 сек.
6	--
7	0.0011218030122108757 сек.
8	--

Вывод по задаче:

Задачи решены с помощью алгоритма, который считывает данные с файла “input.txt”, определяет правильность скобочной последовательности и записывает результат в текстовый файл “output.txt”. Также алгоритм протестирован на время выполнения.

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 находится 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 109. Команда «–» означает изъятие элемента из очереди. Команда «?» означает запрос на поиск минимального элемента в очереди.

• **Формат входного файла (input.txt)**. В первой строке содержится M ($1 \le M \le 10^6$) – число команд. В последующих строках содержатся команды, по одной в каждой строке.

• **Формат выходного файла (output.txt).** Для каждой операции поиска минимума в очереди выведите её результат. Результаты должны быть выведены в том порядке, в котором эти операции встречаются во входном файле. Гарантируется, что операций извлечения или поиска минимума для пустой очереди не производится.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

• Пример:

Input.txt	Output.txt
7 + 1 ? + 10 ? - ? -	1 1 10

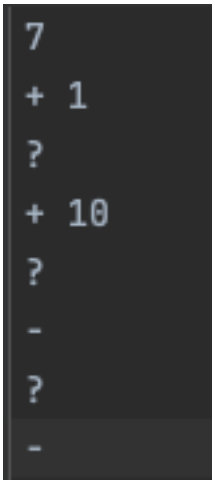
```
f = open('input.txt')
d = [el.replace('\n', '') for el in f.readlines()]
mas = []
o = open('output.txt', 'w')

for cmd in d:
    if cmd[0] == '+':
        mas.append(int(cmd[2:]))
    elif cmd == '-':
        del mas[0]
    elif cmd == '?':
        a = 10**9 + 1
        for i in mas:
            if i < a:
                a = i
        print(a, file=o)
```

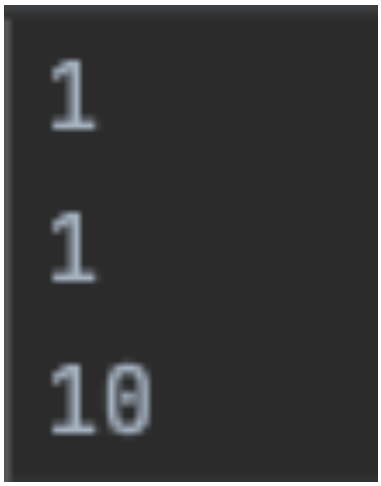
Данная программа решает задачу 6. Сначала открывается входной файл, “input.txt”, затем копируются команды из его строк. Создаем пустой массив, в который будет записываться результат для каждой операции поиска минимума в очереди. Также открываем выходной файл, “output.txt”

Запускаем цикл for. Если команда равна “+”, в пустой массив добавляем число N, если команда содержит “-”, из очереди удаляется элемент, если команда содержит “?”, запускается поиск минимального элемента в очереди.

Поиск минимального элемента в очереди осуществляется следующим образом, создаем какое-либо большое число, сравниваем элемент из очереди с этим числом, если элемент меньше числа, то число меняется на элемент из очереди (следующий элемент из очереди уже будет сравниваться с тем элементом из очереди, а не с большим числом). Записываем минимальный элемент в выходной файл “output.txt”.

Input.txt: Output.txt:





	Время выполнения
Пример из задачи	0.0019466869998723269 сек.

Вывод по задаче:

Задача решена с помощью алгоритма, который считывает данные из входного файла “input.txt”, реализует работу очереди, также отвечает на запрос о минимальном элементе из имеющихся элементов в очереди и, для каждой операции поиска минимума в очереди, выводит ее результат в выходной файл “output.txt”.

Задача №9. “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 ($1 \le n \le 105$) - число запросов к вашей программе. В следующих n строках заданы описания запросов в следующем формате: – «+ i» – к очереди присоединяется пациент i ($1 \le i \le N$) и встает в ее конец; – «* i» – пациент i встает в середину очереди ($1 \le i \le N$); – «-» –

первый пациент в очереди заходит к врачу. Гарантируется, что на момент каждого такого запроса очередь будет не пуста.

• **Формат выходного файла (output.txt).** Для каждого запроса третьего типа в отдельной строке выведите номер пациента, который должен зайти к шаманам.

• Ограничение по времени. Оцените время работы и используемую память при заданных максимальных значениях.

• Пример:

Input.txt	Output.txt		Input.txt	Output.txt
7 + 1 + 2 - + 3 + 4 - -	1 2 3		10 + 1 + 2 * 3 - + 4 * 5 - - - -	1 3 2 5 4

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

class queue:
    def __init__(self):
        self.head = None
        self.tail = None
        self.mid = None
        self.length = 0

    def pop(self):
        if not self.head:
            pass
        host = self.head
        self.head = self.head.next
        if self.length % 2 == 0:
            self.mid = self.mid.next
        self.length -= 1
        return host

    def push(self, value):
        n = Node(value)
```

```
        if self.head == None:
            self.head = n
            self.mid = n
            self.tail = n
            self.length += 1
            return None
        self.tail.next = n
        self.tail = n
        if self.length % 2 == 0:
            self.mid = self.mid.next
        self.length += 1

    def mid_push(self, value):
        n = Node(value)
        if self.head == None:
            self.head = n
            self.mid = n
            self.tail = n
            self.length += 1
            return None
        n.next = self.mid.next
        self.mid.next = n
        if self.length % 2 == 0:
            self.mid = self.mid.next
        self.length += 1

if __name__ == '__main__':
    mas = []
    que = queue()
    with open('input.txt') as f:
        while True:
            l = f.readline()
            if not l:
                break
            mas.append(l.split())
    with open('output.txt', 'w') as o:
    for i in range(int(mas[0][0]) + 1):
        if mas[i][0] == '+':
            que.push(mas[i][1])
        elif mas[i][0] == '*':
            que.mid.push(mas[i][1])
        elif mas[i][0] == '-':
            o.write(f"{que.pop()}\n")
```

input.txt:

```
7
+ 1
+ 2
-
+ 3
+ 4
-
-
```

output.txt:

```
1
2
3
```

	Время выполнения
Пример из задачи	0.0017898880178108811 сек.

Вывод по задаче:

Задача решена с помощью алгоритма, который считывает данные из входного файла “input.txt”, отслеживает порядок пациентов в очереди и, для каждого запроса третьего типа, в отдельной строке выводит в выходной файл “output.txt” номер пациента, который должен зайти к шаманам. Также алгоритм протестирован на время выполнения.