

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 11

Выполнил:
Кузнецов А.Г.
К3140

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Стек	3
Задача №2. Очередь	4
Задача №3. Скобочная последовательность. Версия 1	5
Задача №4. Скобочная последовательность. Версия 2	7
Задача №5. Стек с максимумом	9
Задача №6. Очередь с минимумом	10
Задача №7. Максимум в движущейся последовательности	12
Задача №8. Постфиксная запись	14
Задача №11. Бюрократия	15
Задача №13. Реализация стека, очереди и связанных списков	17
Вывод	23

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 10⁹. Команда “-” означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выполнения команд не превысит 10⁶ 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.

```
1 with open('input.txt', 'r') as f:
2     n = int(f.readline())
3     commands = f.read().splitlines()
4     stack = []
5
6
7 def create_stack(commands, n):
8     for i in range(n):
9         if len(stack) > 10 ** 6:
10             return "Error"
11         if "+" in commands[i]:
12             if int(commands[i][2:]) > 10 ** 9:
13                 return "Error"
14             stack.append(commands[i][2:])
15         else:
16             with open('output.txt', 'a') as f:
17                 f.write(stack.pop() + '\n')
18
19
20 if 1 <= n <= (10 ** 9):
21     create_stack(commands, n)
```

На вход из файла input.txt получаем число команд и сами команды. Проверяем не превышает ли количество команд 10^9 , если число удовлетворяет условию, то запускаем функцию create_stack в которой создаётся наш стек, а изъятые элементы записываются в файл output.txt

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

Вывод по задаче: В ходе работы над первой задачей был реализован процесс нахождения изъятых элементов во время создания стека

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. 2

- Формат входного файла (input.txt). В первой строке содержится M ($1 \leq M \leq 106$) – 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.

```

1 with open('input.txt', 'r') as f:
2     n = int(f.readline())
3     commands = f.read().splitlines()
4     queue = []
5
6
7 def create_queue(commands, n):
8     for i in range(n):
9         if len(queue) > 10 ** 6:
10            return "Error"
11         if "+" in commands[i]:
12             if int(commands[i][2:]) > 10 ** 9:

```

```

13         return "Error"
14         queue.append(commands[i][2:])
15     else:
16         with open('output.txt', 'a') as f:
17             f.write(queue.pop(0) + '\n')
18
19
20 if 1 <= n <= (10 ** 9):
21     create_queue(commands, n)

```

На вход из файла input.txt получаем число команд и сами команды. Проверяем не превышает ли количество команд 10^9 , если число удовлетворяет условию, то запускаем функцию create_queue в которой создаётся наша очередь, а изъятые элементы записываются в файл output.txt

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

Вывод по задаче: В ходе работы над первой задачей был реализован процесс нахождения изъятых элементов во время создания очереди

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 ($1 \leq N \leq 500$) – 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.

```
1 def correct_brackets(A):
2     while '()' in A or '[]' in A:
3         A = A.replace('()', '')
4         A = A.replace('[]', '')
5     if A:
6         return "NO"
7     else:
8         return "YES"
9
10 with open('input.txt', 'r') as f:
11     n = int(f.readline())
12     A = f.read().splitlines()
13     if 1 <= n <= 500:
14         for i in range(n):
15             if 1 <= len(A[i]) <= 10**4:
16                 with open('output.txt', 'a') as f:
17                     f.write(correct_brackets(A[i]) + '\n')
```

На вход из файла input.txt получаем число скобочных последовательностей и сами скобочные последовательности. Проверяем удовлетворяют ли значения условиям, если значения удовлетворяют, то записываем в файл output.txt вывод функции correct_breckets, которая даёт ответ правильная ли скобочная последовательность.

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

Вывод: В ходе выполнения третьей задачи был реализован метод нахождения правильной скобочной последовательности

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 `(` 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 $S - 1 \leq S \leq 105$.

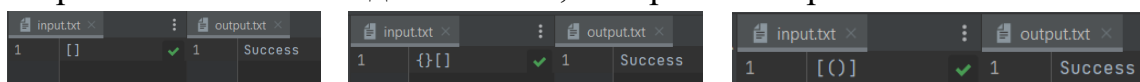
- Формат выходного файла (output.txt). Если в строке S скобки используются правильно, выведите «Success» (без кавычек). В противном случае выведите отсчитываемый от 1 индекс первой несовпадающей закрывающей скобки, а если нет несовпадающих закрывающих скобок, выведите отсчитываемый от 1 индекс первой открывающей скобки, не имеющей закрывающей.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 256 мб.

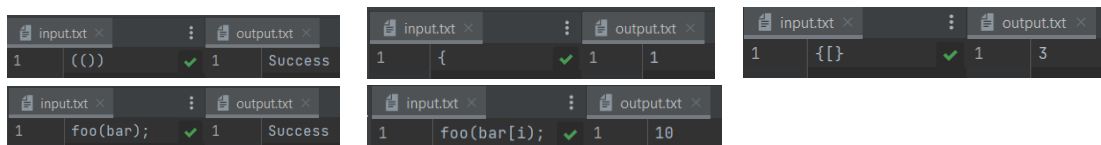
```

1 def correct_brackets(A):
2     k = 1
3     for x in A:
4         if (x == "(" or x == "[" or x == "{"):
5             stack.append(x)
6             num_stack.append(str(k))
7         elif (x == ")" or x == "]" or x == "}"):
8             try:
9                 if not abs(ord(stack.pop()) - ord(x)) <= 2:
10                     return str(k)
11             except IndexError:
12                 return str(k)
13             num_stack.pop()
14
15         k += 1
16     if not num_stack:
17         return "Success"
18     else:
19         return ' '.join(num_stack)
20
21
22 with open('input.txt', 'r') as f:
23     A = f.readline()
24     stack = []
25     num_stack = []
26     if 1 <= len(A) <= 10 ** 5:
27         with open('output.txt', 'w') as f:
28             f.write(correct_brackets(A))

```

На вход из файла input.txt получаем строку, а также создаем два стека: один для скобок, а второй для индексов. Проверяем удовлетворяют ли её длина условию, если длина удовлетворяет, то записываем в файл output.txt вывод функции correct_breckets, ищет в строке скобки и проверяет правильная ли скобочная последовательность. Есть правильная, то функция выводит Success, а если нет, то выводим индекс скобки, которая неправильно закрывается или же индекс скобки, которая не закрылась.





Вывод: В ходе выполнения четвертой задачи был реализован метод нахождения ошибок в построении последовательности скобок в строке и нахождение индекса ошибки

Задача №5. Стек с максимумом

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. $0 \leq V \leq 105$.
- 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

```

1 def push(num, stack):
2     if stack:
3         num_max = max(num, stack[-1][1])
4     else:
5         num_max = num
6     stack.append((num, num_max))
7
8
9 with open('input.txt', 'r') as f:
10     n = int(f.readline())
11     stack = []
12     commands = f.read().splitlines()
13     if 1 <= n <= 400000:
14         for i in range(n):
15             cmd = commands[i].split()
16             if cmd[0] == 'push':
17                 if 0 <= int(cmd[1]) <= 10 ** 5:
18                     push(int(cmd[1]), stack)
19             else:
20                 with open('output.txt', 'w') as f:
21                     f.write("Error")

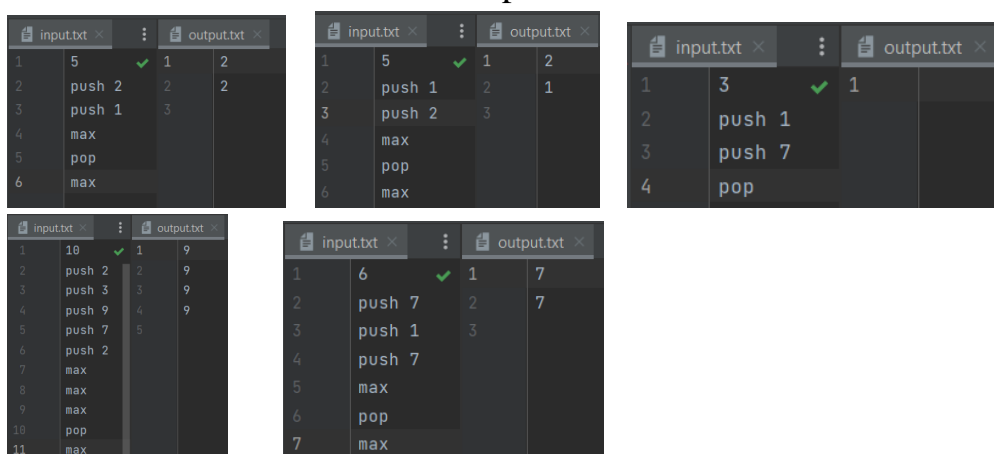
```

```

22         break
23     elif cmd[0] == 'pop':
24         try:
25             stack.pop()
26         except IndexError:
27             with open('output.txt', 'w') as f:
28                 f.write("Error")
29             break
30     elif cmd[0] == 'max':
31         with open('output.txt', 'a') as f:
32             f.write(str(stack[-1][1])+'\n')

```

Из файла input.txt получаем число команд и сами команды, а также создаём стек для чисел. Проверяем удовлетворяет ли число команд условию, если да, то проходимся по командам и проверяем их. При использовании команды push записываем в стек значения с максимальным значением, pop удаляет последний добавленный элемент, а командой max выводим максимальный элемент стека. Полученные данные при использовании команды max записываем в output.txt



Вывод: В ходе выполнения пятой задачи был реализован стек с функцией вывода максимального значения

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 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 М ($1 \leq M \leq 106$) – 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.

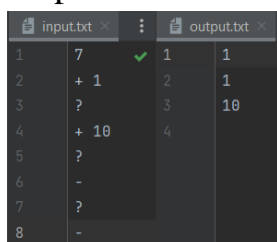
```
1 def push(num, stack, stack_min):
2     if not stack:
3         stack.append(num)
4         stack_min.append(num)
5     else:
6         stack.append(num)
7         while stack_min and stack_min[-1] > num:
8             stack_min.pop()
9             stack_min.append(num)
10
11
12 def pop(stack, stack_min):
13     try:
14         if stack[0] == stack_min[0]:
15             stack.pop(0)
16             stack_min.pop(0)
17         else:
18             stack.pop(0)
19     except IndexError:
20         with open('output.txt', 'w') as f:
21             f.write("Error")
22
23
24 def get_min(stack_min):
25     with open('output.txt', 'a') as f:
26         f.write(str(stack_min[0]) + '\n')
27
28
29 with open('input.txt', 'r') as f:
30     n = int(f.readline())
31     stack = []
32     stack_min = []
33     commands = f.read().splitlines()
34     for i in range(n):
35         cmd = commands[i].split()
36         if cmd[0] == '+':
37             if 0 <= int(cmd[1]) <= 10 ** 9:
```

```

38         push(cmd[1], stack, stack_min)
39     else:
40         with open('output.txt', 'w') as f:
41             f.write("Error")
42             break
43     elif cmd[0] == '-':
44         pop(stack, stack_min)
45     elif cmd[0] == '?':
46         get_min(stack_min)

```

Из файла input.txt получаем число команд и сами команды, а также создаём стек для чисел и стек для минимальных значений. Проверяем удовлетворяет ли число команд условию, если да, то проходимся по командам и проверяем их. При использовании команды push записываем в stack сами значения, а в stack_min минимальные значения, pop удаляет первый добавленный элемент из stack и stack_min, если значения равны иначе удаляется значение только из stack, а командой max выводим максимальный элемент стека. Полученные данные при использовании команды get_min записываем в output.txt



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

Вывод: В ходе выполнения шестой задачи был реализован стек с функцией вывода максимального 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 $O(n)$.

- Формат входного файла (input.txt). В первой строке содержится целое число n ($1 \leq n \leq 105$) – количество чисел в исходном массиве, вторая строка содержит n целых чисел a_1, \dots, a_n этого массива, разделенных пробелом ($0 \leq a_i \leq 105$). В третьей строке - целое число m - ширина "окна" ($1 \leq m \leq n$).

- Формат выходного файла (output.txt). Нужно вывести $\max a_i, \dots, a_{i+m-1}$ для каждого $1 \leq i \leq n - m + 1$.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.

```

1 def get_max(num, stack, stack_max, m, i):
2     if not stack:
3         stack.append(num)
4         stack_max.append(num)
5     else:
6         stack.append(num)
7         while stack_max and stack_max[-1] < num:
8             stack_max.pop()
9         stack_max.append(num)
10    if i >= (m-1):
11        with open('output.txt', 'a') as f:
12            f.write(str(stack_max[0])+' ')
13        try:
14            if stack[0] == stack_max[0]:
15                stack.pop(0)
16                stack_max.pop(0)
17            else:
18                stack.pop(0)
19        except IndexError:
20            with open('output.txt', 'w') as f:
21                f.write("Error")
22
23
24 with open('input.txt', 'r') as f:
25     n = int(f.readline())
26     a = [int(x) for x in f.readline().split(' ')]
27     m = int(f.readline())
28     stack = []
29     stack_max = []
30     if (1 <= n <= 10**5) and (1 <= m <= n):
31         for i in range(n):
32             if 0 <= a[i] <= 10**5:
33                 get_max(a[i], stack, stack_max, m, i)
34             else:
35                 with open('output.txt', 'w') as f:
36                     f.write("Error")
37     else:
38         with open('output.txt', 'w') as f:
39             f.write("Error")

```

Из файла input.txt получаем количество чисел в исходном массиве, сами числа и ширину “окна”. Проверяем удовлетворяют ли значения команд условию, если да, то запускаем функцию get_max, в которой заполняем stack до ширины окна и записываем максимальные значения. Далее выводим максимальные значения в output.txt

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

Вывод: В ходе выполнения седьмой задачи был реализован метод нахождения максимума в движущейся последовательности с помощью двух стеков

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.

- Формат входного файла (input.txt). В первой строке входного файла дано число N ($1 \leq n \leq 106$) – число элементов выражения. Во второй строке содержится выражение в постфиксной записи, состоящее из N элементов. В выражении могут содержаться неотрицательные однозначные числа и операции $+$, $-$, $*$. Каждые два соседних элемента выражения разделены ровно одним пробелом.
- Формат выходного файла (output.txt). Необходимо вывести значение записанного выражения. Гарантируется, что результат выражения, а также результаты всех промежуточных вычислений, по модулю будут меньше, чем 2^{31} .
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

```

1 def command(cmd, num1, num2):
2     if cmd == "*":
3         return num1 * num2
4     elif cmd == "+":
5         return num1 + num2
6     else:
7         return num1 - num2
8
9
10 with open('input.txt', 'r') as f:
11     n = int(f.readline())
12     a = [x for x in f.readline().split(' ')]
13     stack = []

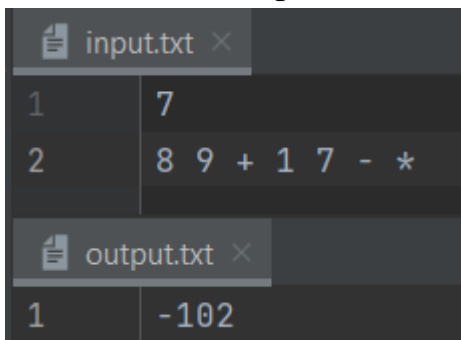
```

```

14 if 1 <= n <= 10 ** 6:
15     for i in range(n):
16         digit = a[i].isdigit()
17         if digit:
18             stack.append(int(a[i]))
19         elif len(stack) >= 2:
20             num2 = stack.pop()
21             num1 = stack.pop()
22             res = command(a[i], num1, num2)
23             stack.append(res)
24         else:
25             with open('output.txt', 'w') as f:
26                 f.write("Error")
27 if len(stack) == 1:
28     with open('output.txt', 'w') as f:
29         f.write(str(stack.pop()))
30 else:
31     with open('output.txt', 'w') as f:
32         f.write("Error")

```

Из файла input.txt получаем число элементов выражения и само выражение в постфиксной записи. Проверяем удовлетворяет ли число элементов выражения условию, если да, то запускаем алгоритм, в котором с помощью стека инициализируем числа и команды и выполняем, непосредственно, выражение. В конце у нас останется стек из одного элемента и его значение записываем в output.txt



Вывод: В ходе выполнения восьмой задачи был реализован алгоритм вычисления постфиксной записи с заданными элементами выражения с помощью стека

Задача №11. Бюрократия

В министерстве бюрократии одно окно для приема граждан. Утром в очередь встают n человек, i -й посетитель хочет получить a_i справок. За один прием можно получить только одну справку, поэтому если после приема

посетителю нужны еще справки, он встает в конец очереди. За время приема министерство успевает выдать m справок. Остальным придется ждать следующего приемного дня. Ваша задача - сказать, сколько еще справок хочет получить каждый из оставшихся в очереди посетитель в тот момент, когда прием закончится. Если все к этому моменту разойдутся, выведите -1.

- Формат входного файла (input.txt). В первой строке - количество посетителей n ($1 \leq n \leq 105$) и количество справок m ($0 \leq m \leq 109$). Во второй строке для каждого посетителя в порядке очереди указано количество справок a_i ($1 \leq a_i \leq 106$), которое он рассчитывает получить. Номером посетителя называется его место в исходной очереди.

- Формат выходного файла (output.txt). В первой строке выведите, сколько посетителей останется в очереди, когда прием закончится. Во второй строке выведите состояние очереди на тот момент, когда прием закончится: для всех посетителей по порядку выведите по одному числу через пробел - количество справок, которое он хочет еще получить. В случае, если в очереди никого не останется выведите одно число: -1

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

```
1 with open('input.txt', 'r') as f:
2     n, m = list(map(int, f.readline().strip().split()))
3     a = [int(x) for x in f.readline().split(' ')]
4     if (1 <= n <= 10 ** 5) and (0 <= m <= 10 ** 9) and (all([1 <= i <=
5 10 ** 6 for i in a])):
6         x = m
7
8         while x != 0:
9             # Пока справок хватает на всю очередь и ни у кого не будет
10            # отрицательного количества справок после раздачи
11            while (x - (len(a)) >= 0) and (all([i > 0 for i in a])):
12                k = 1
13                while x - ((k+1) * len(a)) >= 0:
14                    k+=1
15                x -= len(a)*k
16                a = [x - k for x in a]
17                while a.count(0) != 0:
18                    a.remove(0)
19                if a[0] - 1 == 0 and x > 0:
20                    a.pop(0)
21                    x -= 1
22                elif a[0] - 1 != 0 and x > 0:
23                    a[0]-=1
24                    a.append(a.pop(0))
25                    x -= 1
26            if len(a) == 0:
```



```

27         with open('output.txt', 'w') as f:
28             f.write("-1")
29     else:
30         with open('output.txt', 'w') as f:
31             f.write(str(len(a)))
32             f.write("\n")
33             f.write(' '.join(map(str, a)))

```

Из файла input.txt получаем количество посетителей, справок и количество справок для каждого посетителя. Проверяем удовлетворяют ли значения условию, если да, то вычитаем единицу от количества билетов у каждого посетителя пока справок хватает на всю очередь и ни у кого не отрицательного количества справок после раздачи. После чего проходимся по оставшемуся массиву и вычитаем количество требуемых справок у посетителей до тех пор, пока количество справок не станет равным нулю. Далее ответ записываем в output.txt

input.txt	
1	3 2
2	1 2 3
output.txt	
1	2
2	3 1

input.txt	
1	4 5
2	2 5 2 3
output.txt	
1	3
2	4 1 2

input.txt	
1	100000 1000000000
2	855771 122722 436124 594551 661170 213918 346167 770881 195440 117422 972831 240369 121676 427751 715343 387331 783074 638001 595694 55244 597845 715174 770859 923744 355912 798949 586637 893098 376135 924363 317358 163813 36393 384100 253252 205407 60064 000742 445545 715547
output.txt	
1	100000
2	845771 112722 426124 584551 651170 203918 336167 760881 185440 107422 962831 230369 111676 417751 705343 377331 773074 628001 585694 45244 587845 705174 760859 913744 345912 788949 576637 883098 366135 914363 307358 153813 26393 374100 243252

	Время выполнения	Затраты памяти
Задача №11	0.08360719999473076 секунд	0.002992 Мб

Вывод: В ходе выполнения одиннадцатой задачи был реализован метод нахождения количества посетителей в очереди и состояние очереди с помощью циклов

Задача №13. Реализация стека, очереди и связанных списков

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

```

1 # Класс стека с функциями isEmpty, push, pop и вывода данных
2 class Stack:
3

```

```

4      # Класс для создания узла связанного списка,
5      # где первый элемент значение, а второй ссылка на второй элемент
6      class Node:
7          def __init__(self, data, next=None):
8              self.data = data
9              self.next = next
10
11      def __init__(self):
12          self.top = None
13
14      # Функция для добавления элемента в стек
15      def push(self, data): # вставить в начало
16
17          # Выделяет новый узел
18          node = Stack.Node(data)
19
20          # устанавливает данные в выделенном узле
21          node.data = data
22
23          # устанавливает указатель .next нового узла так, чтобы он
24 указывал на текущий
25          # верхний узел списка
26          node.next = self.top
27
28          # обновить верхний указатель
29          self.top = node
30
31
32
33      # Функция для извлечения из стека верхнего значения и его удаления
34 из стека
35      def pop(self):
36          if self.isEmpty():
37              print('Error pop')
38              exit(-1)
39
40          # Принимает к сведению данные верхнего узла
41          top = self.top.data
42
43          # Обновляет верхний указатель, чтобы он указывал на следующий
44 узел
45          self.top = self.top.next
46
47          return top
48
49      # Функция вывода верхнего значения без последующего его удаления
50      def peek(self):
51          if not self.isEmpty():
52              return self.top.data
53          else:
54              print('Error peek')
55              exit(-1)
56
57      # Функция для проверки пуст ли стек или нет
58      def isEmpty(self):
59          return self.top is None

```

```

60
61
62
63 if __name__ == '__main__':
64
65     stack = Stack()
66
67     stack.push(2)
68     stack.push(7)
69     stack.push(3)
70     stack.push(1)
71     stack.push(5)
72     print('Верхний элемент', stack.peek())
73     stack.push(2)
74     stack.push(6)
75     stack.push(2)
76     print('Верхний элемент', stack.peek())
77
78     stack.pop()
79     stack.pop()
80     stack.pop()
81     stack.pop()
82     stack.pop()
83     stack.pop()
84     print('Верхний элемент', stack.peek())
85     stack.pop()
86     stack.pop()
87
88     if stack.isEmpty():
89         print('Стек пустой')
90     else:
91         print('Стек не пустой')

```

```

stack.pop()

```

```

Верхний элемент 5
Верхний элемент 2
Верхний элемент 7
Стек пустой
Error pop

```

Вывод: В ходе выполнения первого пункта тринадцатой задачи был реализован стек на основе связанного списка с функциями isEmpty, push, pop и вывода данных

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

```

1 # Класс очереди с функциями Enqueue, Dequeue с проверкой
2 # на переполнение и опустошения очереди.
3 class Queue:
4
5     # Класс для создания узла связанного списка,
6     # где первый элемент значение, а второй ссылка на второй элемент

```

```

7     class Node:
8         def __init__(self, data):
9             # устанавливает данные в выделенном узле и возвращает их
10            self.data = data
11            self.next = None
12
13        def __init__(self):
14            self.back = None
15            self.front = None
16
17        # Функция для добавления элемента в очередь
18        def enqueue(self, item): # Вставка # в конце
19
20            # Выделяет новый узел
21            node = Queue.Node(item)
22
23            # Проверка пуста ли очередь
24            if self.front is None:
25                # Инициализирует как верхние, так и нижние
26                self.front = node
27                self.back = node
28            else:
29                # Обновление сзади
30                self.back.next = node
31                self.back = node
32
33        # Функция для извлечения из очереди нижнего значения и его удаления
34        из очереди
35        def dequeue(self):
36            if self.front is None:
37                print('Очередь опустошена')
38                exit(-1)
39
40            # Принимает к сведению данные верхнего узла
41            front = self.front.data
42
43            # Обновляет верхний указатель, чтобы он указывал на следующий
44            узел
45            self.front = self.front.next
46
47            # Если список станет пустым
48            if self.front is None:
49                self.back = None
50
51            # вернуть удаленный элемент
52            return front
53
54        # Функция вывода нижнего значения без последующего его удаления
55        def peek(self):
56            if not self.isEmpty():
57                return self.front.data
58            else:
59                print('Error peek')
60                exit(-1)
61
62        # Функция для проверки пуста ли очередь или нет

```

```

63     def isEmpty(self):
64         return self.back is None and self.front is None
65
66
67 if __name__ == '__main__':
68
69     q = Queue()
70
71     q.enqueue(2)
72     q.enqueue(7)
73     q.enqueue(3)
74     q.enqueue(1)
75     q.enqueue(5)
76     print('Нижний элемент', q.peek())
77     q.enqueue(2)
78     q.enqueue(6)
79     q.enqueue(2)
80     print('Нижний элемент', q.peek())
81
82     q.dequeue()
83     q.dequeue()
84     q.dequeue()
85     q.dequeue()
86     q.dequeue()
87     q.dequeue()
88     print('Нижний элемент', q.peek())
89     q.dequeue()
90     q.dequeue()
91
92     if q.isEmpty():
93         print('Очередь пуста')
94     else:
95         print('Очередь не пуста')

```

q.dequeue()

```

Нижний элемент 2
Нижний элемент 2
Нижний элемент 6
Очередь пуста
Очередь опустошена

```

Вывод: В ходе выполнения второго пункта тринадцатой задачи был реализована очередь на основе связанного 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

Вывод

В ходе лабораторной работы была проведена работа с функционалом стека и очереди