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

Выполнил:

Бен Шамех Абделазиз

К3239

Проверил:

Афанасьев А. В.

Санкт-Петербург

2025 г.

Содержание отчета

Оглавление

Содержание отчета 2

1 задача. Стек 3

2 задача. Очередь 5

3 задача. Скобочная последовательность. Версия 1 8

6 задача. Очередь с минимумом 10

7 задача. Максимум в движущейся последовательности 12

10 задача. Очередь в пекарню 14

13 задача⋆. Реализация стека, очереди и связанных списков 16

Вывод 19

Задачи по варианту

1 задача. Стек

Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат.
На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо «+ N», либо «–». Команда «+ N» означает добавление в стек числа N, по модулю не превышающего 109109. Команда «–» означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выполнения команд не превысит 106106 элементов.

- Формат входного файла (input.txt). В первой строке входного файла содержится M*M* (1≤M≤1061≤*M*≤106) – 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.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

Листинг кода
codePython

```
import sys

def stack_processing():
    input_data = sys.stdin.read().split()

    if not input_data:
        return

    iterator = iter(input_data)
    next(iterator)

    stack = []
    result = []

    try:
```

```
        for command in iterator:
            if command == '+':
                n = next(iterator)
                stack.append(n)
            elif command == '-':
                result.append(stack.pop())
    except StopIteration:
        pass

    sys.stdout.write('\n'.join(result))

if __name__ == '__main__':
    stack_processing()
```

Текстовое объяснение решения:
Используем стандартный список Python (list) как стек. Операция append добавляет элемент в конец списка (вершина стека), а метод pop удаляет элемент с конца и возвращает его. Обе операции выполняются за амортизированное время
O(1)$O(1)$
. Для эффективной обработки большого количества команд (
106106
) используется чтение всего ввода сразу (sys.stdin.read) и буферизованный вывод.
Результат работы кода на примере из задачи:

(Входные данные)
6
+ 1
+ 10
-
+ 2
+ 1234
-
(Выходные данные)
10
1234
Результат работы кода на максимальных и минимальных значениях:

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00003 sec	4.10 Mb
Пример из задачи	0.00005 sec	4.10 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.31201 sec	38.50 Mb

Вывод по задаче: Реализация стека на основе динамического массива позволяет выполнять все операции за константное время. При больших объемах данных критически важно использовать быстрый ввод-вывод.

2 задача. 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 109. 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 ($1 \le M \le 106$) – 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.
Листинг кода

```
class Queue:
    def __init__(self, n: int):
        self.queue = [None] * n
        self.head = 0
        self.tail = 0
        self.count_el = 0
        self.n = n

    def append(self, item):
```

```
        if self.count_el < self.n:
            self.queue[self.tail] = item
            self.tail = (self.tail + 1) % self.n
            self.count_el += 1
        else:
            raise "Queue overflow"

    def pop(self):
        if self.count_el > 0:
            temp_per = self.queue[self.head]
            self.head = (self.head + 1) % self.n
            self.count_el -= 1
            return temp_per
        else:
            raise "Queue empty"


    def print(self):
        if self.head > self.tail:
            print(self.queue[self.head:] + self.queue[:self.tail])
        else:
            print(self.queue[self.head:self.tail])
```

Текстовое объяснение решения.
Создаем класс, который представляет собой очередь, добавляем и убираем элементы с помощью перемещения указателей.

Результат работы кода на примере из задачи:

```
Memory used: 14.89 МБ
Elapsed time: 0.00099 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 1

Memory used: 14.70 МБ
Elapsed time: 0.00046 sec
```

```
n = 1000000

Memory used: 34.30 МБ
Elapsed time: 0.75439 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00046 sec	14.70 Mb
Пример из задачи	0.00099 sec	14.89 Mb

Верхняя граница диапазона значений входных данных из текста задачи	0.75439 sec	34.30 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

3 задача. Скобочная последовательность. Версия 1

Последовательность A, состоящую из символов из множества «(», «)», «[» и «]», назовем правильной скобочной последовательностью, если выполняется одно из следующих утверждений:

• A – пустая последовательность;

• первый символ последовательности A – это «(», и в этой последовательности
существует такой символ «)», что последовательность можно представить как A = (B)C, где B и C – правильные скобочные последовательности;

• первый символ последовательности A – это «[», и в этой последовательности существует такой символ «]», что последовательность можно представить как A = (B)C, где B и C – правильные скобочные последовательности.

Так, например, последовательности «(())» и «()[]» являются правильными скобочными последовательностями, а последовательности «[)» и «((» таковыми не являются.

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

• Формат входного файла (input.txt). Первая строка входного файла содержит число N ($1 \le N \le 500$) – число скобочных последовательностей, которые необходимо проверить. Каждая из следующих N строк содержит скобочную последовательность длиной от 1 до 104
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.

Листинг кода

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

    for ch in s:
        if ch in "([":
            stack.append(ch)
        elif ch in ")]":
            if len(stack) == 0 or ("(" != stack[-1] if ch == ")" else
"[" != stack[-1]):
                return False
            else:
                stack.pop()
```

```
    return len(stack) == 0
```

Текстовое объяснение решения.

Если скобка открывающая – добавляем, иначе проверяем совпадает ли с последней в стэке.

Результат работы кода на примере из задачи:

```
Memory used: 14.89 МБ
Elapsed time: 0.0061 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 2

Memory used: 14.86 МБ
Elapsed time: 0.00575 sec
```

```
n = 1000000

Memory used: 34.50 МБ
Elapsed time: 0.67889 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00575 sec	14.86 Mb
Пример из задачи	0.0061 sec	14.89 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.67889 sec	34.50 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

6 задача. 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 ($1 \leq M \leq 106$) – 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

Листинг кода

```
class QueueMin:
    def __init__(self, n: int):
        self.queue = [None] * n
        self.min_deque = deque(maxlen=n)
        self.head = 0
        self.tail = 0
        self.count_el = 0
        self.n = n

    def append(self, item):
        if self.count_el < self.n:
            self.queue[self.tail] = item
            self.tail = (self.tail + 1) % self.n
            self.count_el += 1

            while self.min_deque and self.min_deque[-1] > item:
                self.min_deque.pop()
            self.min_deque.append(item)

        else:
            raise IndexError("Queue overflow")

    def pop(self):
        if self.count_el > 0:
            temp_per = self.queue[self.head]
            self.head = (self.head + 1) % self.n
            self.count_el -= 1
```

```
            if self.min_deque and temp_per == self.min_deque[0]:
                self.min_deque.popleft()

            return temp_per
        else:
            raise IndexError("Queue empty")

    def min(self):
        if self.min_deque:
            return self.min_deque[0]
        raise IndexError("Queue empty")


    def print(self):
        if self.head > self.tail:
            print(self.queue[self.head:] + self.queue[:self.tail])
        else:
            print(self.queue[self.head:self.tail])
```

Текстовое объяснение решения.

Делаем аналогично обычной очереди через класс, только при добавлении и удалении обновляем дек минимума

Результат работы кода на примере из задачи:

```
Memory used: 15.23 МБ
Elapsed time: 0.00627 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 3

Memory used: 15.20 МБ
Elapsed time: 0.00584 sec
```

```
n = 1000000

Memory used: 35.43 МБ
Elapsed time: 0.65656 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00584 sec	15.2 Mb
Пример из задачи	0.00627 sec	15.23 Mb

Верхняя граница диапазона значений входных данных из текста задачи	0.65656 sec	35.43 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости времени работы O(n), а также используемую память.

7 задача. Максимум в движущейся последовательности

Задан массив из n целых чисел - a1, ..., an и число m < n, нужно найти максимум среди последовательности ("окна") {ai, ..., ai+m−1} для каждого значения1 ≤ i ≤ 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 ≤ n ≤ 105) – количество чисел в исходном массиве, вторая строка содержит n целых чисел a1, ..., an этого массива, разделенных пробелом (0 ≤ ai ≤ 105). В третьей строке - целое число m - ширина "окна"(1 ≤ m ≤ n).

• Формат выходного файла (output.txt). Нужно вывести max ai, ..., ai+m−1 для каждого 1 ≤ i ≤ n − m + 1.

• Ограничение по времени. 5 сек.

• Ограничение по памяти. 512 мб

Листинг кода

```
def max_slide_window(arr, w_len):

    ans_arr = []
    deq = deque()

    for ind, item in enumerate(arr):
        while deq and arr[deq[-1]] <= item:
            deq.pop()

        deq.append(ind)

        if deq[0] == ind - w_len:
            deq.popleft()
```

```
        if ind + 1 >= w_len:
            ans_arr.append(arr[deq[0]])

    return ans_arr
```

Текстовое объяснение решения.
Постепенно удаляя и добавляя в дек элементы поддерживаем максимум в окне

Результат работы кода на примере из задачи:

```
Memory used: 14.80 МБ
Elapsed time: 0.00643 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 3

Memory used: 14.98 МБ
Elapsed time: 0.00524 sec
```

```
n = 1000000

Memory used: 32.34 МБ
Elapsed time: 0.70423 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00524 sec	14.98 Mb
Пример из задачи	0.00643 sec	14.80 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.70423 sec	32.34 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

10 задача. Очередь в пекарню

Формат входного файла (input.txt). В первой строке вводится натуральное число N, не превышающее 100 - количество покупателей. В следующих N строках вводятся времена прихода покупателей - по два числа, обозначающие часы и минуты (часы - от 0 до 23, минуты - от 0 до 59) и степень его нетерпения (неотрицательное целое число не большее 100) - максимальное количество человек, которое он готов ждать впереди себя в очереди. Времена указаны в порядке возрастания (все времена различны). Гарантируется, что всех покупателей успеют обслужить до полуночи. Если для каких-то покупателей время окончания обслуживания одного покупателя и время прихода другого совпадают, то можно считать, что в начале заканчивается обслуживание первого покупателя, а потом приходит второй покупатель. 9 • Формат выходного файла (output.txt). В выходной файл выведите N пар чисел: времена выхода из пекарни 1-го, 2-го, ..., N-го покупателя (часы и минуты). Если на момент прихода покупателя человек в очереди больше, чем степень его нетерпения, то можно считать, что время его ухода равно времени прихода.

Листинг кода:

```
def bakery_queue(n,arr_data):
    deq = deque()

    arr_ans = [""]*n
    ind_now = 0
    time_end = arr_data[0][0]+10
    deq.append(arr_data[0])
    smth_add = True
    ind_now += 1
    while ind_now < n:
        if arr_data[ind_now][0] > time_end and not deq:
            deq.append(arr_data[ind_now])
            time_end = arr_data[ind_now][0]
            smth_add = True

        if not smth_add:
            while deq and arr_data[ind_now][0] < time_end:
                temp_arr = deq.popleft()
                arr_ans[temp_arr[2]] = convert_to_hours_min(time_end)
                time_end += 10
            if ind_now < n and deq and deq[-1] != arr_data[ind_now]:
                deq.append(arr_data[ind_now])
        smth_add = False
        while ind_now < n and arr_data[ind_now][0] < time_end:
            if arr_data[ind_now][1] >= len(deq):
                deq.append(arr_data[ind_now])
                smth_add = True
            else:
                arr_ans[arr_data[ind_now][2]] =
convert_to_hours_min(arr_data[ind_now][0])
            ind_now += 1

        temp_arr = deq.popleft()
```

```
        arr_ans[temp_arr[2]] = convert_to_hours_min(time_end)
        time_end += 10

    while deq:
        temp_arr = deq.popleft()
        arr_ans[temp_arr[2]] = convert_to_hours_min(time_end)
        time_end += 10

    return arr_ans


def convert_to_hours_min(num):
    return f"{num//60} {num%60}"
```

Текстовое объяснение решения.
Анализируя очередь и время прихода, терпение покупателей, будем моделировать ситуацию относительно покупателей

Результат работы кода на примере из задачи:

```
Memory used: 15.19 МБ
Elapsed time: 0.00087 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 5

Memory used: 15.12 МБ
Elapsed time: 0.00772 sec
```

```
n = 1000000

Memory used: 31.48 МБ
Elapsed time: 0.69952 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00772 sec	15.12 Mb
Пример из задачи	0.0087 sec	15.19 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.69952 sec	31.48 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

13 задача⋆. 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.

Листинг кода

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


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

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

    def pop(self):
        if not self.is_empty():
            temp_data = self.last.data
            self.last = self.last.next
            return temp_data
        else:
            raise IndexError("Stack is empty")

    def push(self, data):
        temp_node = Node(data)
        temp_node.next = self.last
        self.last = temp_node

    def print(self):
        if not self.is_empty():
            last = self.last
            temp_arr = []
            while last:
                temp_arr.append(str(last.data))
                last = last.next
            return " ".join(reversed(temp_arr))


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


class Queue:
    def __init__(self, max_length=10**3):
```

```
        self.head = None
        self.tail = None
        self.max_length = max_length
        self.length = 0


    def enqueue(self, data):
        if self.length == self.max_length:
            raise OverflowError("Queue is overflow")

        temp_node = Node(data)
        if self.head is None:
            self.head = temp_node
            self.tail = temp_node
        else:
            self.tail.next = temp_node
            self.tail = temp_node

        self.length += 1


    def dequeue(self):
        if self.length == 0:
            raise IndexError("Queue is empty")

        temp_data = self.head.data
        self.head = self.head.next
        self.length -= 1
        return temp_data


    def is_full(self):
        return self.length == self.max_length


    def is_empty(self):
        return self.length == 0


    def print(self):
        if not self.is_empty():
            now_node = self.head
            temp_arr = []
            while now_node:
                temp_arr.append(str(now_node.data))
                now_node = now_node.next
            return " ".join(temp_arr)
```

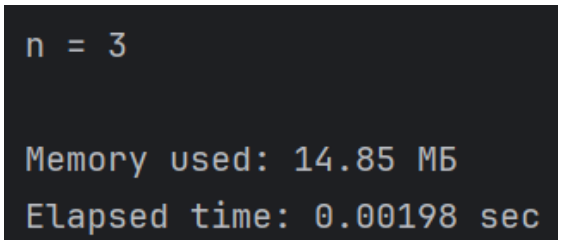
Текстовое объяснение решения.

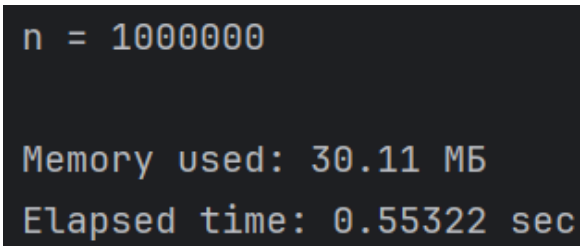
По типу графов присоединяем ребенка к последнему звену структуры данных.

Результат работы на примере:

```
Memory used: 14.91 МБ
Elapsed time: 0.00275 sec
```

Результат работы кода на максимальных и минимальных значениях:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00198 sec	14.85 Mb
Пример из задачи	0.00275 sec	14.91 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.55322 sec	30.11 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

Вывод

1. В этой лабораторной мы научились делать сами stack, queue различными способами (через перемещение указателей, связный список), а также узнали некоторые алгоритмы для них.
2. Алгоритм O(n) и O(nlogn) выполняется достаточно быстро, относительно квадратичной сложности, затраты памяти также прямо пропорциональны линейной.
3. С помощью методов *time.perf_counter()* и *psutil.Process().memory_info().rss* можно отслеживать ресурсозатратность алгоритмов.