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
Группа: K3239

Проверила:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание

1 Задача 1. Стек **2**
1.1 Условие . . . 2
1.2 Листинг кода . . . 2
1.3 Текстовое объяснение . . . 2
1.4 Результаты выполнения . . . 2

2 Задача 3. Скобочная последовательность. Версия 1 **3**
2.1 Условие . . . 3
2.2 Листинг кода . . . 3
2.3 Текстовое объяснение . . . 3
2.4 Результаты выполнения . . . 3

3 Задача 6. Очередь с минимумом **4**
3.1 Листинг кода . . . 4
3.2 Текстовое объяснение . . . 4
3.3 Результаты выполнения . . . 4

4 Задача 7. Максимум в движущейся последовательности **5**
4.1 Листинг кода . . . 5
4.2 Текстовое объяснение . . . 5
4.3 Результаты выполнения . . . 5

5 Задача 10. Очередь в пекарню **6**
5.1 Условие . . . 6
5.2 Листинг кода . . . 6
5.3 Текстовое объяснение . . . 6
5.4 Результаты выполнения . . . 7

6 Задача 13. Стек и Очередь на связных списках **8**
6.1 Листинг кода . . . 8
6.2 Результаты выполнения . . . 8

7 Вывод **9**

1 Задача 1. Стек

1.1 Условие

Реализуйте работу стека. Команды: «+ N» (добавить число) и «-» (извлечь и вывести верхнее число).

1.2 Листинг кода

```python
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 comm in iterator:
            if comm == '+':
                stack.append(next(iterator))
            elif comm == '-':
                result.append(stack.pop())
    except StopIteration: pass
    sys.stdout.write('\n'.join(result))
```

1.3 Текстовое объяснение

Используем стандартный список Python как стек. Операция `append` добавляет элемент на вершину, а `pop` удаляет и возвращает последний элемент. Обе операции работают за амортизированное время $O(1)$.

1.4 Результаты выполнения

Сценарий	**Время (сек)**	**Память (Mb)**
Пример из задачи	0.00005 sec	4.10 Mb
Верхняя граница (n=10^6)	0.31201 sec	38.50 Mb

2 Задача 3. Скобочная последовательность. Версия 1

2.1 Условие

Проверить, является ли последовательность скобок вида «()» и «[]» правильной.

2.2 Листинг кода

```python
def valid_brackets(s):
    stack = []
    for ch in s:
        if ch in "([":
            stack.append(ch)
        elif ch in ")]":
            if not stack: return False
            top = stack.pop()
            if (ch == ")" and top != "(") or (ch == "]" and top != "[")
   :
                return False
    return len(stack) == 0
```

2.3 Текстовое объяснение

Используем стек для сопоставления открывающих и закрывающих скобок. При встрече закрывающей скобки она должна соответствовать верхней скобке в стеке. Если стек пуст или типы не совпадают — последовательность неверна.

2.4 Результаты выполнения

Сценарий	**Время (сек)**	**Память (Mb)**
Пример из задачи	0.0061 sec	14.89 Mb
Верхняя граница	0.67889 sec	34.50 Mb

3 Задача 6. Очередь с минимумом

3.1 Листинг кода

```python
from collections import deque

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

    def pop(self):
        temp_per = self.queue[self.head]
        if self.min_deque and temp_per == self.min_deque[0]:
            self.min_deque.popleft()
        self.head = (self.head + 1) % self.n
        self.count_el -= 1
        return temp_per

    def min(self): return self.min_deque[0]
```

3.2 Текстовое объяснение

Для получения минимума за $O(1)$ используется вспомогательный монотонный дек (`min_deque`). При добавлении элемента удаляются все элементы в конце дека, которые больше текущего. Минимум всегда находится в голове дека.

3.3 Результаты выполнения

Сценарий	Время (сек)	Память (Mb)
Пример из задачи	0.00627 sec	15.23 Mb
Верхняя граница	0.65656 sec	35.43 Mb

4 Задача 7. Максимум в движущейся последовательности

4.1 Листинг кода

```python
from collections import deque

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
```

4.2 Текстовое объяснение

Реализован алгоритм за $O(n)$ с использованием индексированного монотонного дека. Дек хранит индексы элементов окна в порядке убывания их значений.

4.3 Результаты выполнения

Сценарий	**Время (сек)**	**Память (Mb)**
Пример из задачи	0.00643 sec	14.80 Mb
Верхняя граница	0.70423 sec	32.34 Mb

5 Задача 10. Очередь в пекарню

5.1 Условие

Моделирование работы пекарни. Покупатель уходит, если в момент его прихода очередь длиннее его степени нетерпения. Обслуживание занимает 10 минут.

5.2 Листинг кода

```
from collections import deque

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

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])
    ind_now += 1

    while ind_now < n or deq:
        if not deq and ind_now < n:
            time_end = arr_data[ind_now][0]

        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])
            else:
                arr_ans[arr_data[ind_now][2]] = convert_to_hours_min(
  arr_data[ind_now][0])
            ind_now += 1

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

5.3 Текстовое объяснение

Моделируем обслуживание покупателей. Если покупатель приходит и видит, что количество людей в очереди (`len(deq)`) превышает его терпение, он уходит сразу (время выхода = время прихода). Иначе он встает в `deque` и обслуживается по 10 минут в порядке очереди.

5.4 Результаты выполнения

Сценарий	**Время (сек)**	**Память (Mb)**
Нижняя граница (n=5)	0.00772 sec	15.12 Mb
Пример из задачи	0.0087 sec	15.19 Mb
Верхняя граница	0.69952 sec	31.48 Mb

6 Задача 13. Стек и Очередь на связных списках

6.1 Листинг кода

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

class Stack:
    def __init__(self):
        self.last = None
    def push(self, data):
        new_node = Node(data)
        new_node.next = self.last
        self.last = new_node
    def pop(self):
        if self.last:
            val = self.last.data
            self.last = self.last.next
            return val
```

6.2 Результаты выполнения

Сценарий	**Время (сек)**	**Память (Mb)**
Пример из задачи	0.00275 sec	14.91 Mb
Верхняя граница	0.55322 sec	30.11 Mb

7 Вывод

1. В данной лабораторной работе были изучены и реализованы базовые структуры данных: стек, очередь и связанные списки.

2. Использование монотонных деков позволило оптимизировать задачи поиска экстремумов в скользящем окне до линейного времени $O(n)$.

3. Реализации через массивы и связные списки показали высокую эффективность, позволяя обрабатывать миллионы запросов в секунду.