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

1.1 Условие

Реализуйте работу стека. На вход подаются команды «+ N» (добавление) или «-» (извлечение).

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

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

1.3 Объяснение решения

Используем стандартный список Python (list) как стек. Операция `append` добавляет элемент в конец, а `pop` удаляет и возвращает последний элемент. Оба метода работают за амортизированное время $O(1)$.

1.4 Результаты выполнения

Пример из задачи:

- **Input:** `6, + 1, + 10, -, + 2, + 123, -`
- **Output:** `10, 123`

Метрики эффективности:

Сценарий	Время (сек)	Память (Mb)
Нижняя граница	0.00003 sec	4.10 Mb
Пример из задачи	0.00005 sec	4.10 Mb
Верхняя граница	0.31201 sec	38.50 Mb

2 Задача 2. Очередь

2.1 Листинг кода

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

2.2 Результаты выполнения

Сценарий	**Время (сек)**	**Память (Mb)**
Нижняя граница	0.00046 sec	14.70 Mb
Пример из задачи	0.00099 sec	14.89 Mb
Верхняя граница	0.75439 sec	34.30 Mb

3 Задача 3. Скобочная последовательность. Версия 1

3.1 Листинг кода

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

3.2 Результаты выполнения

Сценарий	Время (сек)	Память (Mb)
Нижняя граница	0.00575 sec	14.86 Mb
Пример из задачи	0.0061 sec	14.89 Mb
Верхняя граница	0.67889 sec	34.50 Mb

4 Задача 6. Очередь с минимумом

4.1 Листинг кода

```python
from collections import deque

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

4.2 Результаты выполнения

Сценарий	Время (сек)	Память (Mb)
Нижняя граница	0.00584 sec	15.2 Mb
Пример из задачи	0.00627 sec	15.23 Mb
Верхняя граница	0.65656 sec	35.43 Mb

5 Задача 7. Максимум в движущейся последовательности

5.1 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
```

5.2 Результаты выполнения

Сценарий	**Время (сек)**	**Память (Mb)**
Нижняя граница	0.00524 sec	14.98 Mb
Пример из задачи	0.00643 sec	14.80 Mb
Верхняя граница	0.70423 sec	32.34 Mb

6 Задача 10. Очередь в пекарню

6.1 Листинг кода

```python
from collections import deque

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]] = f"{arr_data[ind_now][0]//60} {arr_data[ind_now][0]%60}"
            ind_now += 1

        if deq:
            customer = deq.popleft()
            arr_ans[customer[2]] = f"{time_end//60} {time_end%60}"
            time_end += 10
    return arr_ans
```

6.2 Результаты выполнения

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

7 Задача 13. Реализация стека, очереди и связанных списков

7.1 Листинг кода

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

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

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

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

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

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
```

7.2 Результаты выполнения

Сценарий	**Время (сек)**	**Память (Mb)**
Нижняя граница	0.00198 sec	14.85 Mb
Пример из задачи	0.00275 sec	14.91 Mb
Верхняя граница	0.55322 sec	30.11 Mb

8 Вывод

1. В данной лабораторной работе мы изучили реализацию таких структур данных, как стек (*stack*) и очередь (*queue*), различными способами: через динамические массивы и через узлы связанных списков.

2. Алгоритмы со сложностью $O(n)$ и $O(1)$ для базовых операций позволяют обрабатывать миллионы запросов за доли секунды.

3. Использование специализированных методов (дек для минимума) позволяет эффективно решать задачи скользящего окна.