МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Пермский государственный национальный
исследовательский университет»

Институт компьютерных наук и
технологий

ОТЧЁТ

по индивидуальной работе №2
по дисциплине «Язык программирования Python»
Вариант 11

Работу выполнил
студент группы ИТ/О ИТ-1-2024 1 курса
Рябов Сергей Александрович
«18» июня 2025 г.

Работу проверил
Анисимова С.И.
«___» июня 2025 г.

Пермь 2025

СОДЕРЖАНИЕ

Постановка задачи

Баржа. На ней находится K отсеков. В каждый можно поместить некоторое количество бочек с одним из 10000 видов топлива. Извлечь бочку из отсека можно лишь в случае, если все бочки, помещённые в этот отсек после неё, уже были извлечены. В каждый момент времени в каждом непустом отсеке имеется ровно одна бочка, которую можно извлечь, не трогая остальных. Изначально баржа пуста. Затем она последовательно проплывает через N доков, причём в каждом доке на баржу либо погружается бочка с некоторым видом топлива в некоторый отсек, либо выгружается крайняя бочка из некоторого отсека. Однако, если указанный отсек пуст, либо если выгруженная бочка содержит не тот вид топлива, который ожидалось, следует зафиксировать ошибку. Если на баржу оказывается погружено более P бочек или если после прохождения всех доков она не стала пуста, следует также зафиксировать ошибку. Нужно найти максимальное количество бочек, которые одновременно пребывали на барже либо зафиксировать ошибку.

Алгоритм решения

Инициализация структуры данных "Стек":

Класс Stack - класс, реализующий отсеки в Барже

Методы:

1. push() - используем для добавления бочки в отсек
2. pop() - используем для извлечения бочки из отсека
3. is_empty() - используем для проверки отсека на пустоту
4. __str__() - используем для строкового представления отсека

Используем стек потомучто бочки в барже погружаются и разгружаются по принципу LIFO (last in first out)

Инициализация класса "Баржа":

Класс Barge - основной класс, реализующий логику работы баржи:

1. _holds - список списков (стеков) для хранения бочек в каждом отсеке
2. _current - текущее количество бочек
3. _max - максимальное достигнутое количество бочек
4. _limit - лимит бочек

5. _error - флаг ошибки
6. _error_message - сообщение об ошибке

Методы класса:

1. add_barrel() - добавляет бочку в указанный отсек
2. remove_barrel() - извлекает бочку из указанного отсека
3. is_empty() - проверяет пустоту баржи
4. get_state() - возвращает текущее состояние
5. get_result() - возвращает итоговый результат

Основная часть программы:

Пользователь вводит данные (K, P, N) пользователем. После чего выбирает режим: автоматический или ручной. В автоматическом с помощью функции generate_operations() создаются и выводятся операции. В ручном пользователь последовательно вводит каждую операцию. После получения списка операций программа создаёт Баржу и начинает обработку. После всех операций выполняет финальную проверку на ошибки, если они отсутствуют то программа выводит ответ.

Тестирование

```
Введите количество отсеков (K >= 1): 1
Введите лимит бочек (P >= 0 и P <= 100000): 5
Введите количество операций (N >= 1): 4

Выберите режим:
1 - Автоматическая генерация операций
2 - Ручной ввод операций
> 1

Сгенерированные операции:
+ 1 9347 (добавляем 9347 в отсек 1)
+ 1 8994 (добавляем 8994 в отсек 1)
- 1 8994 (выгружаем 8994 из отсека 1)
+ 1 7721 (добавляем 7721 в отсек 1)
```

```
Начальное состояние:
Отсек 1: []
Текущее количество: 0
Максимальное количество: 0
Ошибка: нет

Шаг 1: Добавление в отсеке 1 (топливо: 9347)
Отсек 1: [9347]
Текущее количество: 1
Максимальное количество: 1
Ошибка: нет

Шаг 2: Добавление в отсеке 1 (топливо: 8994)
Отсек 1: [9347, 8994]
Текущее количество: 2
Максимальное количество: 2
Ошибка: нет

Шаг 3: Извлечение в отсеке 1 (топливо: 8994)
Отсек 1: [9347]
Текущее количество: 1
Максимальное количество: 2
Ошибка: нет

Шаг 4: Добавление в отсеке 1 (топливо: 7721)
Отсек 1: [9347, 7721]
Текущее количество: 2
Максимальное количество: 2
Ошибка: нет
```

```
! Обнаружена ошибка: ОШИБКА: Баржа не пуста в конце

=== Результат работы ===
Error
```

Первый тест, итог - ошибка

```
Введите количество отсеков (K >= 1): 1
Введите лимит бочек (P >= 0 и P <= 100000): 1
Введите количество операций (N >= 1): 2

Выберите режим:
1 - Автоматическая генерация операций
2 - Ручной ввод операций
> 1

Сгенерированные операции:
+ 1 5991 (добавляем 5991 в отсек 1)
+ 1 7536 (добавляем 7536 в отсек 1)
```

```
Начальное состояние:
Отсек 1: []
Текущее количество: 0
Максимальное количество: 0
Ошибка: нет

Шаг 1: Добавление в отсеке 1 (топливо: 5991)
Отсек 1: [5991]
Текущее количество: 1
Максимальное количество: 1
Ошибка: нет

Шаг 2: Добавление в отсеке 1 (топливо: 7536)
Отсек 1: [5991, 7536]
Текущее количество: 2
Максимальное количество: 2
Ошибка: ОШИБКА: ОШИБКА: Превышен лимит 1 бочек
! Остановка: ОШИБКА: ОШИБКА: Превышен лимит 1 бочек

=== Результат работы ===
Error
```

Второй тест, итог - ошибка

```
Введите количество отсеков (K >= 1): 2
Введите лимит бочек (P >= 0 и P <= 100000): 3
Введите количество операций (N >= 1): 2

Выберите режим:
1 - Автоматическая генерация операций
2 - Ручной ввод операций
> 1

Сгенерированные операции:
+ 2 3744 (добавляем 3744 в отсек 2)
- 2 3744 (выгружаем 3744 из отсека 2)
Начальное состояние:
Отсек 1: []
Отсек 2: []
Текущее количество: 0
Максимальное количество: 0
Ошибка: нет

Шаг 1: Добавление в отсеке 2 (топливо: 3744)
Отсек 1: []
Отсек 2: [3744]
Текущее количество: 1
Максимальное количество: 1
Ошибка: нет

Шаг 2: Извлечение в отсеке 2 (топливо: 3744)
Отсек 1: []
Отсек 2: []
Текущее количество: 0
Максимальное количество: 1
Ошибка: нет

=== Результат работы ===
1
```

Третий тест, итог - 1

Код программы

```
import random


class Stack:
    def __init__(self):
        self._items = []

    def push(self, item):
        """Добавляет элемент на вершину стека"""
        self._items.append(item)

    def pop(self):
        """Удаляет и возвращает элемент с вершины стека"""
        if self.is_empty():
            raise IndexError("Попытка извлечь из пустого стека")
        return self._items.pop()

    def is_empty(self):
        """Проверяет, пуст ли стек"""
        return len(self._items) == 0

    def __str__(self):
        """Строковое представление стека"""
        return str(self._items)


class Barge:
    def __init__(self, num_holds, max_barrels):
        self._holds = [Stack() for _ in range(num_holds + 1)]  #
Индексация отсеков с 1
        self._current = 0  # Текущее количество бочек
        self._max = 0  # Максимальное количество бочек
        self._limit = max_barrels  # Лимит бочек
        self._error = False  # Флаг ошибки
        self._error_message = ""  # Сообщение об ошибки

    def _validate_hold(self, hold_num):
        """Проверка номера отсека (приватный метод)"""
        if not 1 <= hold_num < len(self._holds):
            self._error = True
            self._error_message = f"ОШИБКА: Неверный номер
отсека {hold_num}"
            return False
        return True

    def _check_limit(self):
        """Проверка лимита бочек (приватный метод)"""
        if self._current > self._limit:
            self._error = True
```

```
            self._error_message = f"ОШИБКА: Превышен лимит
{self._limit} бочек"
            raise ValueError(self._error_message)

    def _update_max(self):
        """Обновление максимального количества (приватный
метод)"""
        if self._current > self._max:
            self._max = self._current

    def add_barrel(self, hold_num, fuel_type):
        """Добавление бочки в отсек (публичный метод)"""
        if self._error:
            return False

        if not self._validate_hold(hold_num):
            return False

        try:
            self._holds[hold_num].push(fuel_type)
            self._current += 1
            self._update_max()
            self._check_limit()
            return True
        except Exception as e:
            self._error = True
            self._error_message = f"ОШИБКА: {str(e)}"
            return False

    def remove_barrel(self, hold_num, expected_fuel):
        """Извлечение бочки из отсека (публичный метод)"""
        if self._error:
            return False

        if not self._validate_hold(hold_num):
            return False

        try:
            if self._holds[hold_num].is_empty():
                raise ValueError(f"Отсек {hold_num} пуст")

            actual_fuel = self._holds[hold_num].pop()
            if actual_fuel != expected_fuel:
                raise ValueError(f"Ожидалось {expected_fuel}, а
получили {actual_fuel}")

            self._current -= 1
            return True
        except Exception as e:
            self._error = True
            self._error_message = f"ОШИБКА: {str(e)}"
            return False
```

```
    def is_empty(self):
        """Проверка, пуста ли баржа (публичный метод)"""
        if self._error:
            return False
        return all(hold.is_empty() for hold in self._holds[1:])

    def get_state(self):
        """Возвращает строку с текущим состоянием (публичный
метод)"""
        state = []
        for i, hold in enumerate(self._holds[1:], 1):
            state.append(f"Отсек {i}: {hold}")
        state.append(f"Текущее количество: {self._current}")
        state.append(f"Максимальное количество: {self._max}")
        state.append(f"Ошибка:      {self._error_message      if
self._error else 'нет'}")
        return "\n".join(state)

    def get_result(self):
        """Финальный результат (публичный метод)"""
        if not self.is_empty() and not self._error:
            self._error = True
            self._error_message  =  "ОШИБКА:  Баржа  не  пуста  в
конце"
        return self._max if not self._error else "Error"


def generate_operations(N, K):
    """Генерация случайных операций"""
    operations = []
    hold_states = [[] for _ in range(K + 1)]

    print("\nСгенерированные операции:")
    for _ in range(N):
        available  =  [i  for  i  in  range(1,  K  +  1)  if
hold_states[i]]

        if available and random.random() < 0.5:
            hold = random.choice(available)
            fuel = hold_states[hold][-1]
            operations.append(('-', hold, fuel))
            print(f"- {hold} {fuel} (выгружаем {fuel} из отсека
{hold})")
            hold_states[hold].pop()
        else:
            hold = random.randint(1, K)
            fuel = random.randint(1, 10000)
            operations.append(('+', hold, fuel))
            print(f"+ {hold} {fuel} (добавляем {fuel} в отсек
{hold})")
            hold_states[hold].append(fuel)
```

```
    return operations


def manual_input_operations(N, K):
    """Ручной ввод операций"""
    operations = []
    print("\nВводите операции в формате: [+/-] [номер_отсека]
[тип_топлива]")
    print("Пример: + 1 5000 - добавить бочку 5000 в отсек 1")
    print("        - 2 3000 - извлечь бочку 3000 из отсека 2\n")

    for i in range(1, N + 1):
        while True:
            op_str = input(f"Операция {i}/{N}: ").strip()
            parts = op_str.split()
            if len(parts) != 3:
                print("Ошибка: введите 3 значения через пробел!")
                continue

            op, hold_str, fuel_str = parts
            if op not in ('+', '-'):
                print("Ошибка: операция должна быть + или -!")
                continue

            try:
                hold = int(hold_str)
                fuel = int(fuel_str)
                if not 1 <= hold <= K:
                    print(f"Ошибка: номер отсека должен быть от
1 до {K}!")
                    continue
                if fuel <= 0:
                    print("Ошибка:  тип  топлива  должен  быть
положительным числом!")
                    continue
                break
            except ValueError:
                print("Ошибка: номер отсека и тип топлива должны
быть целыми числами!")

        operations.append((op, hold, fuel))
    return operations


def simulate():
    """Основная функция симуляции"""
    while True:
        try:
            K = int(input("Введите количество отсеков (K >= 1):
"))
            if K < 1:
```

```
                print("Ошибка: количество отсеков должно быть не
менее 1!")
                continue

            P = int(input("Введите лимит бочек (P >= 0 и P <=
100000): "))
            if P < 0 or P > 100000:
                print("Ошибка: лимит бочек должен быть от 0 до
100000!")
                continue

            N = int(input("Введите количество операций (N >= 1):
"))
            if N < 1:
                print("Ошибка: количество операций должно быть
не менее 1!")
                continue
            break
        except ValueError:
            print("Ошибка: введите целые числа!")

    # Выбор режима
    while True:
        mode = input("\nВыберите режим:\n1 - Автоматическая
генерация операций\n2 - Ручной ввод операций\n> ").strip()
        if mode in ('1', '2'):
            break
        print("Ошибка: введите 1 или 2")

    # Генерация операций
    operations = generate_operations(N, K) if mode == '1' else
manual_input_operations(N, K)

    # Инициализация баржи
    barge = Barge(K, P)
    print("\nНачальное состояние:")
    print(barge.get_state())

    # Обработка операций
    for i, (op, hold, fuel) in enumerate(operations, 1):
        print(f"\nШаг {i}: {'Добавление' if op == '+' else
'Извлечение'} в отсеке {hold} (топливо: {fuel})")

        success = barge.add_barrel(hold, fuel) if op == '+' else
barge.remove_barrel(hold, fuel)
        print(barge.get_state())

        if not success:
            print(f"! Остановка: {barge._error_message}")
            break

    # Финальная проверка
```

```
    if not barge._error and not barge.is_empty():
        barge._error = True
        barge._error_message = "ОШИБКА: Баржа не пуста в конце"
        print(f"\n! Обнаружена ошибка: {barge._error_message}")

    # Результат
    print("\n=== Результат работы ===")
    print(barge.get_result())


if __name__ == "__main__":
    simulate()
```

https://github.com/Knuck16/ikm_2sem.git