|  |  |  |
| --- | --- | --- |
| 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 3](#_Toc153130027)

[Алгоритм решения 3](#_Toc153130028)

[Тестирование](#_Toc153130029) 5

[Код программы](#_Toc153130030) 8

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

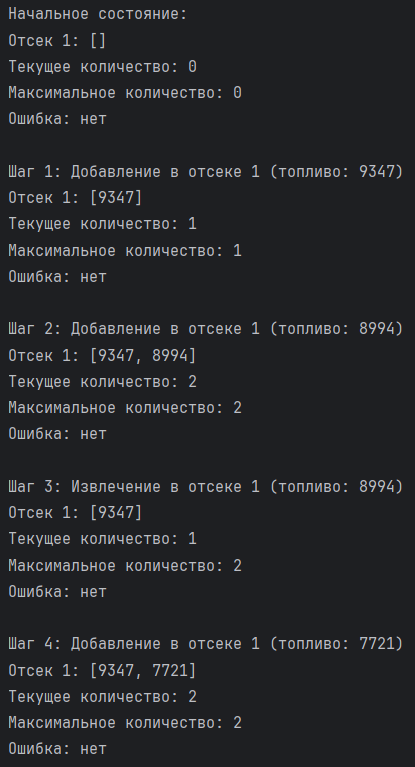
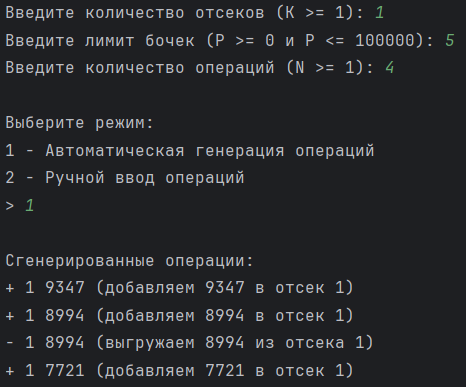
Методы класса:

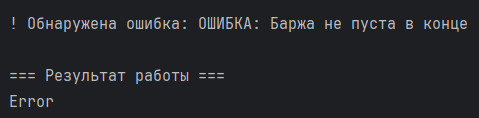
1. add\_barrel() - добавляет бочку в указанный отсек
2. remove\_barrel() - извлекает бочку из указанного отсека
3. is\_empty() - проверяет пустоту баржи
4. get\_state() - возвращает текущее состояние
5. get\_result() - возвращает итоговый результат

Основная часть программы:

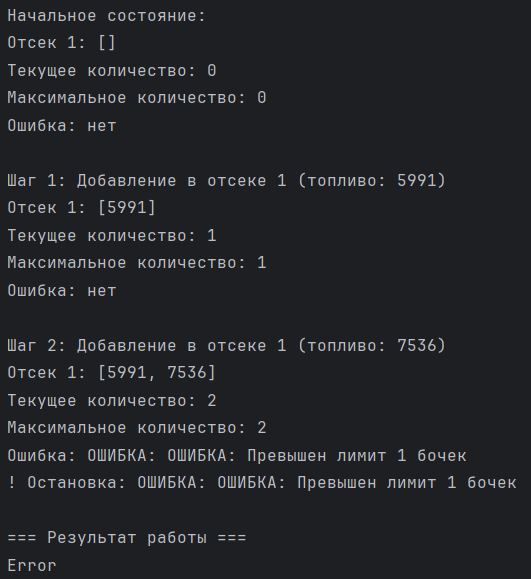
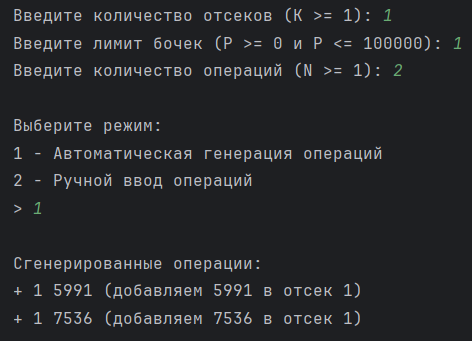
Пользователь вводит данные (K, P, N) пользователем. После чего выбирает режим: автоматический или ручной. В автоматическом с помощью функции generate\_operations() создаются и выводятся операции. В ручном пользователь последовательно вводит каждую операцию. После получения списка операций программа создаёт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

Тестирование

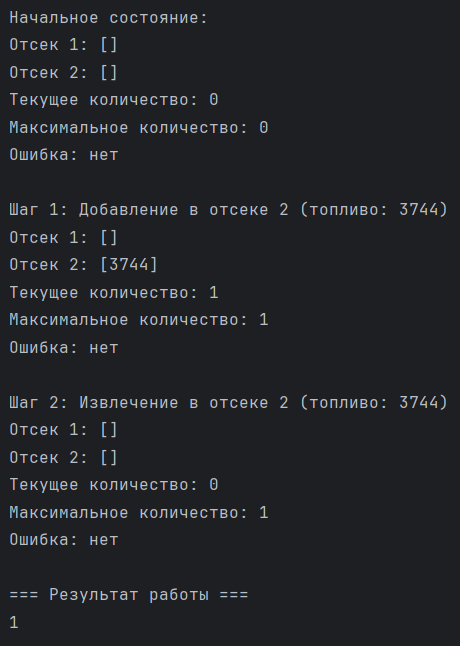
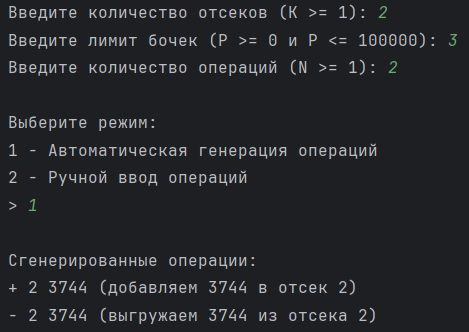




Первый тест, итог - ошибка



Второй тест, итог - ошибка



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>