САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Алгоритмы и структуры данных»

Тема: Жадные алгоритмы. Динамическое программирование

Вариант 20

Выполнил:

Смирнов Георгий Валерьевич

К3139

Проверила:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[**Содержание отчета 2**](#_gjdgxs)

[**Задачи по варианту 3**](#_30j0zll)

[Задача №6. [0.5 баллов] 3](#_1fob9te)

[Задача №7. [0.5 баллов] 6](#_3znysh7)

[Задача №8. [1 баллов] 8](#_2et92p0)

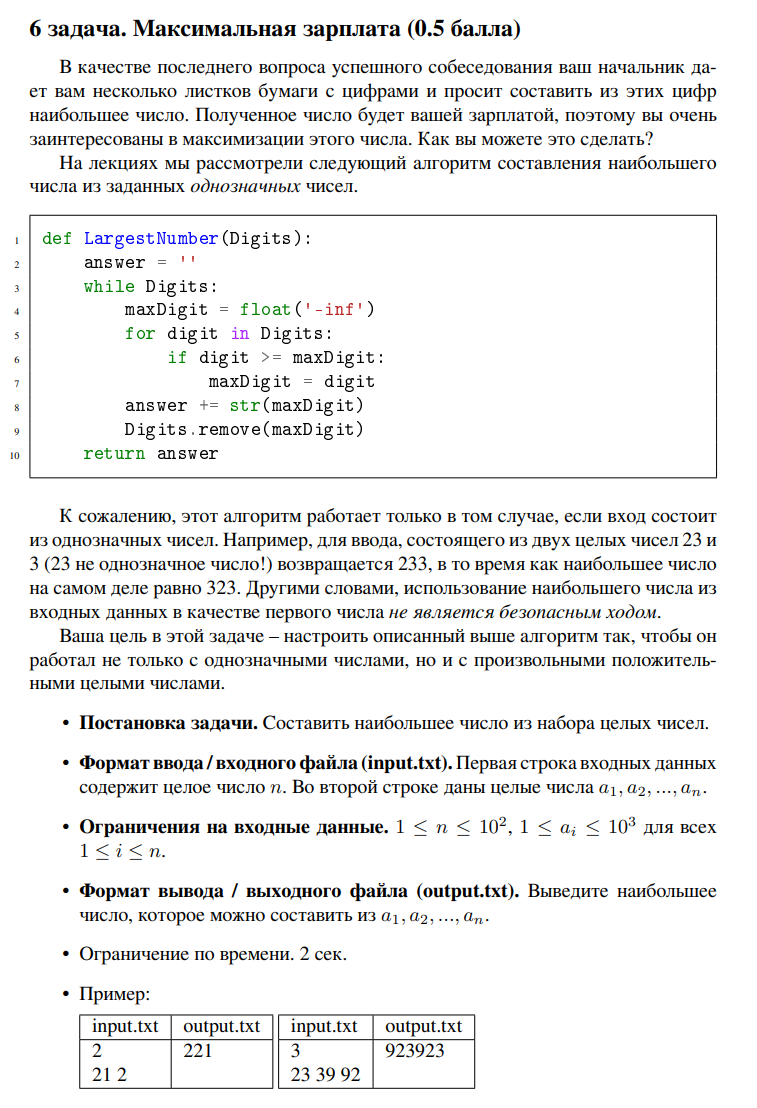
[Задача №13. [2 баллов] 10](#_tyjcwt)

[Задача №21. [3 баллов] 12](#_3dy6vkm)

[**Вывод 23**](#_26in1rg)

# Задачи по варианту

## Задача №6. [0.5 баллов]



**Код программы**

import time

def largest\_number(numbers):

from functools import cmp\_to\_key

# Функция сравнения для сортировки

def compare(x, y):

if x + y > y + x:

return -1

elif x + y < y + x:

return 1

else:

return 0

# Преобразуем числа в строки

numbers = list(map(str, numbers))

# Сортируем числа по нашему правилу

numbers.sort(key=cmp\_to\_key(compare))

# Объединяем отсортированные числа в одно

largest\_num = ''.join(numbers)

# Убираем ведущие нули (если есть)

return largest\_num.lstrip('0') or '0'

# Запуск таймера

start\_time = time.time()

# Чтение входных данных из файла

with open('input.txt', 'r') as file:

lines = file.readlines()

# Проверка на наличие хотя бы двух строк в файле

if len(lines) < 2:

raise ValueError("Файл input.txt должен содержать как минимум две строки: количество чисел и сами числа.")

# Чтение количества чисел

n = int(lines[0].strip())

# Проверка, что количество чисел больше нуля и не превышает 100

if n == 0:

raise ValueError("Количество чисел должно быть больше нуля.")

if n > 100:

raise ValueError("Количество чисел не должно превышать 100.")

# Чтение чисел и преобразование их в список целых чисел

numbers = list(map(int, lines[1].strip().split()))

# Проверка, что количество чисел соответствует указанному значению n

if len(numbers) != n:

raise ValueError("Количество чисел не соответствует указанному значению.")

# Проверка, что все числа в диапазоне от 1 до 1000

for number in numbers:

if number < 1 or number > 1000:

raise ValueError(f"Число {number} выходит за пределы допустимого диапазона (1-1000).")

# Получение наибольшего числа

result = largest\_number(numbers)

# Запись результата в файл

with open('output.txt', 'w') as file:

file.write(result)

# Остановка таймера и вывод времени выполнения

end\_time = time.time()

execution\_time = end\_time - start\_time

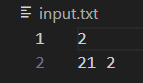
print(f"Время выполнения: {execution\_time:.6f} секунд")

**Текстовое объяснение решения**

После считывания данных, сортируем числа по специальному правилу: если конкатенация двух чисел в одном порядке больше, чем в другом, то первое число должно идти перед вторым. После сортировки объединяем отсортированные строки в одно число и убираем ведущие нули, если они есть.

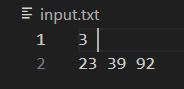
Результат работы кода на примерах из текста задачи:

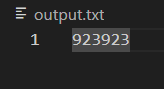
1)





2)

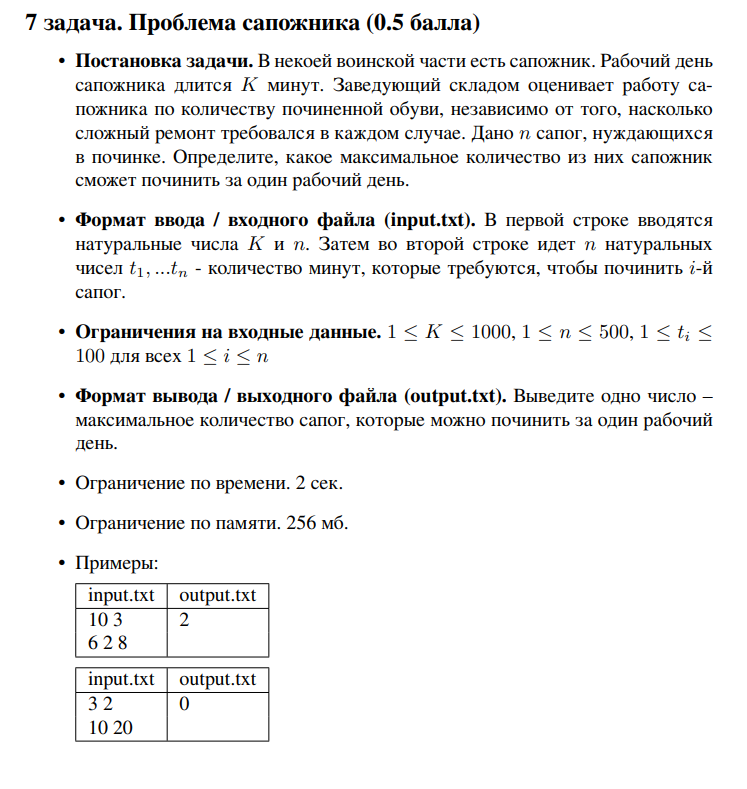




|  | Время выполнения, с |
| --- | --- |
| Пример из задачи | 0.007991 секунд |
| Пример из задачи | 0.012847 секунд |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

## Задача №7. [0.5 баллов]



**Код программы**

import time

# Запуск таймера

start\_time = time.time()

# Чтение входных данных из файла

with open('input.txt', 'r') as file:

lines = file.readlines()

# Проверка на наличие хотя бы двух строк в файле

if len(lines) < 2:

raise ValueError("Файл input.txt должен содержать как минимум две строки: количество минут и количество сапог, а также время починки каждого сапога.")

# Чтение количества минут и количества сапог

K, n = map(int, lines[0].strip().split())

# Проверка ограничений на K и n

if not (1 <= K <= 1000):

raise ValueError("Количество минут K должно быть в диапазоне от 1 до 1000.")

if not (1 <= n <= 500):

raise ValueError("Количество сапог n должно быть в диапазоне от 1 до 500.")

# Чтение времени починки каждого сапога

times = list(map(int, lines[1].strip().split()))

# Проверка, что количество времени починки соответствует количеству сапог

if len(times) != n:

raise ValueError("Количество времени починки не соответствует количеству сапог.")

# Проверка ограничений на время починки каждого сапога

for t in times:

if not (1 <= t <= 100):

raise ValueError(f"Время починки {t} выходит за пределы допустимого диапазона (1-100).")

# Сортировка времени починки в порядке возрастания

times.sort()

# Подсчет максимального количества сапог, которые можно починить

total\_time = 0

count = 0

for t in times:

if total\_time + t <= K:

total\_time += t

count += 1

else:

break

# Запись результата в файл

with open('output.txt', 'w') as file:

file.write(str(count))

# Остановка таймера и вывод времени выполнения

end\_time = time.time()

execution\_time = end\_time - start\_time

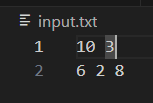
print(f"Время выполнения: {execution\_time:.6f} секунд")

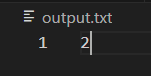
**Текстовое объяснение решения**

После считывания данных сортируем время починки в порядке возрастания. После этого подсчитываем максимальное количество сапог, которые можно починить за K минут. Для этого проходим по отсортированному списку времени починки и суммируем время, пока общая сумма не превысит K. Если добавление следующего времени починки превышает K, прекращаем подсчет.

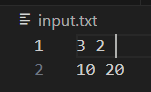
Результат работы кода на примерах из текста задачи:

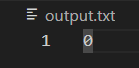
1)





2)

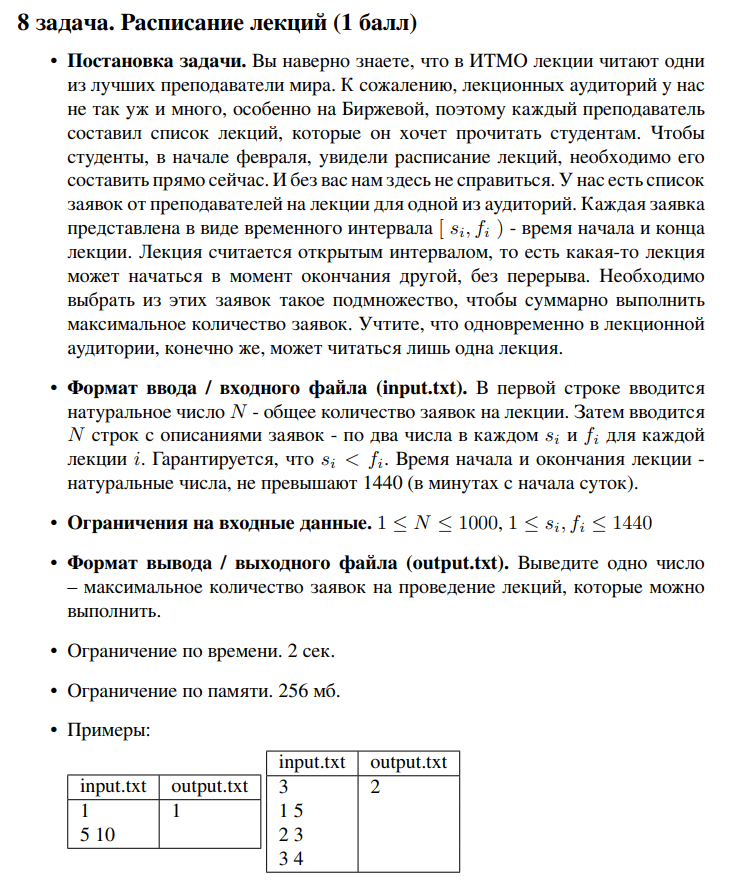




|  | Время выполнения, с |
| --- | --- |
| Пример из задачи | 0.000578 секунд |
| Пример из задачи | 0.001415 секунд |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

## Задача №8. [1 баллов]



**Код программы**

import time

import psutil

import os

# Запуск таймера

start\_time = time.time()

# Чтение входных данных из файла

with open('input.txt', 'r') as file:

lines = file.readlines()

# Проверка на наличие хотя бы одной строки в файле

if len(lines) < 1:

raise ValueError("Файл input.txt должен содержать как минимум одну строку: количество заявок.")

# Чтение количества заявок

N = int(lines[0].strip())

# Проверка ограничений на N

if not (1 <= N <= 1000):

raise ValueError("Количество заявок N должно быть в диапазоне от 1 до 1000.")

# Чтение временных интервалов для каждой заявки

intervals = []

for i in range(1, N + 1):

s, f = map(int, lines[i].strip().split())

# Проверка ограничений на s и f

if not (1 <= s < f <= 1440):

raise ValueError(f"Временные интервалы {s} и {f} выходят за пределы допустимого диапазона (1-1440).")

intervals.append((s, f))

# Сортировка заявок по времени окончания

intervals.sort(key=lambda x: x[1])

# Выбор непересекающихся заявок

count = 0

last\_end\_time = 0

for s, f in intervals:

if s >= last\_end\_time:

last\_end\_time = f

count += 1

# Запись результата в файл

with open('output.txt', 'w') as file:

file.write(str(count))

# Остановка таймера и вывод времени выполнения

end\_time = time.time()

execution\_time = end\_time - start\_time

print(f"Время выполнения: {execution\_time:.6f} секунд")

# Проверка использования памяти

process = psutil.Process(os.getpid())

memory\_usage = process.memory\_info().rss / (1024 \* 1024)

print(f"Использование памяти: {memory\_usage:.2f} МБ")

# Проверка ограничения по памяти

if memory\_usage > 256:

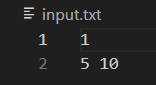
raise MemoryError("Использование памяти превысило 256 МБ")

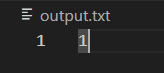
**Текстовое объяснение решения**

После считывания данных сортируем заявки по времени окончания. После этого выбираем максимальное количество непересекающихся заявок. Для этого проходим по отсортированному списку заявок и добавляем заявку в выбор, если её начало не пересекается с концом последней выбранной заявки.

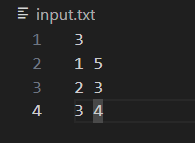
Результат работы кода на примерах из текста задачи:

1)





2)

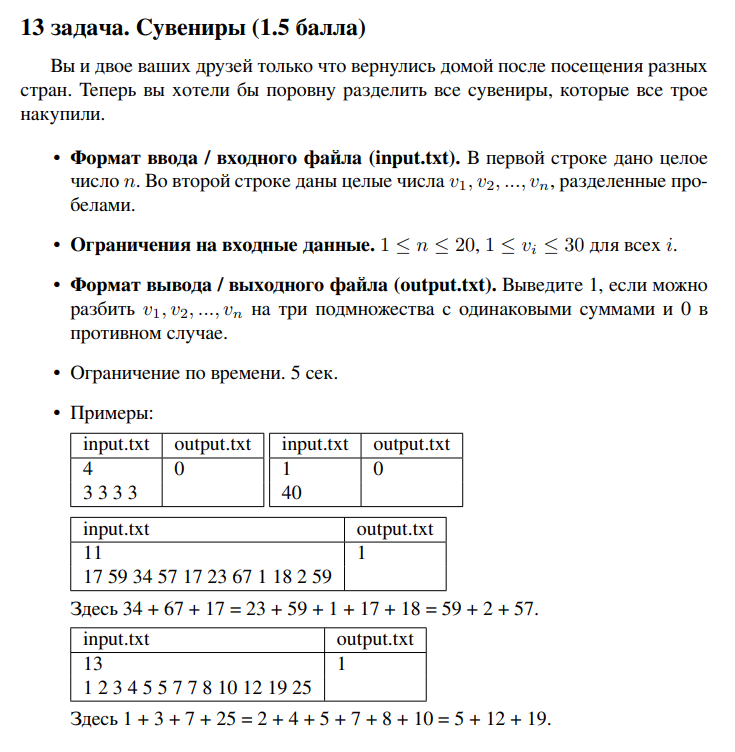




|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.001186899986 секунд | 15.02 МБ |
| Пример из задачи | 0.000850699987 секунд | 15.05 МБ |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

## Задача №13. [2 баллов]



**Код программы**

import time

import os

def can\_partition\_into\_three\_subsets(nums):

total\_sum = sum(nums)

# Если сумма не делится на 3, то невозможно разделить на три подмножества с одинаковыми суммами

if total\_sum % 3 != 0:

return 0

target = total\_sum // 3

n = len(nums)

# dp[i][j] будет True, если можно получить сумму j, используя первые i элементов

dp = [[False] \* (target + 1) for \_ in range(n + 1)]

dp[0][0] = True

for i in range(1, n + 1):

for j in range(target, -1, -1):

if j >= nums[i-1]:

dp[i][j] = dp[i][j] or dp[i-1][j-nums[i-1]]

dp[i][j] = dp[i][j] or dp[i-1][j]

return 1 if dp[n][target] else 0

# Запуск таймера

start\_time = time.time()

# Чтение входных данных из файла

with open('input.txt', 'r') as file:

lines = file.readlines()

# Проверка на наличие хотя бы двух строк в файле

if len(lines) < 2:

raise ValueError("Файл input.txt должен содержать как минимум две строки: количество чисел и сами числа.")

# Чтение количества чисел

n = int(lines[0].strip())

# Проверка, что количество чисел больше нуля и не превышает 20

# if n == 0:

# raise ValueError("Количество чисел должно быть больше нуля.")

# if n > 20:

# raise ValueError("Количество чисел не должно превышать 20.")

# Чтение чисел и преобразование их в список целых чисел

nums = list(map(int, lines[1].strip().split()))

# Проверка, что количество чисел соответствует указанному значению n

# if len(nums) != n:

# raise ValueError("Количество чисел не соответствует указанному значению.")

# Проверка, что все числа в диапазоне от 1 до 30

# for num in nums:

# if num < 1 or num > 30:

# raise ValueError(f"Число {num} выходит за пределы допустимого диапазона (1-30).")

# Проверка возможности разделить на три подмножества с одинаковыми суммами

result = can\_partition\_into\_three\_subsets(nums)

# Запись результата в файл

with open('output.txt', 'w') as file:

file.write(str(result))

# Остановка таймера и вывод времени выполнения

end\_time = time.time()

execution\_time = end\_time - start\_time

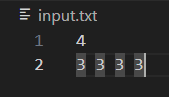
print(f"Время выполнения: {execution\_time:.6f} секунд")

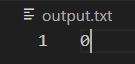
**Текстовое объяснение решения**

После считывания данных Создаем двумерный массив dp, где dp[количество элементов][сумма] будет True, если можно получить сумму, используя первое количество элементов. Инициализируем dp[0][0] как True, так как сумма 0 всегда достижима без использования элементов. Затем заполняем массив dp, проверяя для каждого элемента, можно ли включить его в подмножество для достижения нужной суммы.

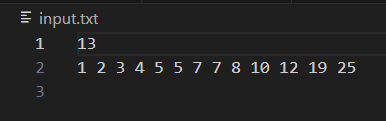
Результат работы кода на примерах из текста задачи:

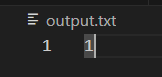
1)





2)

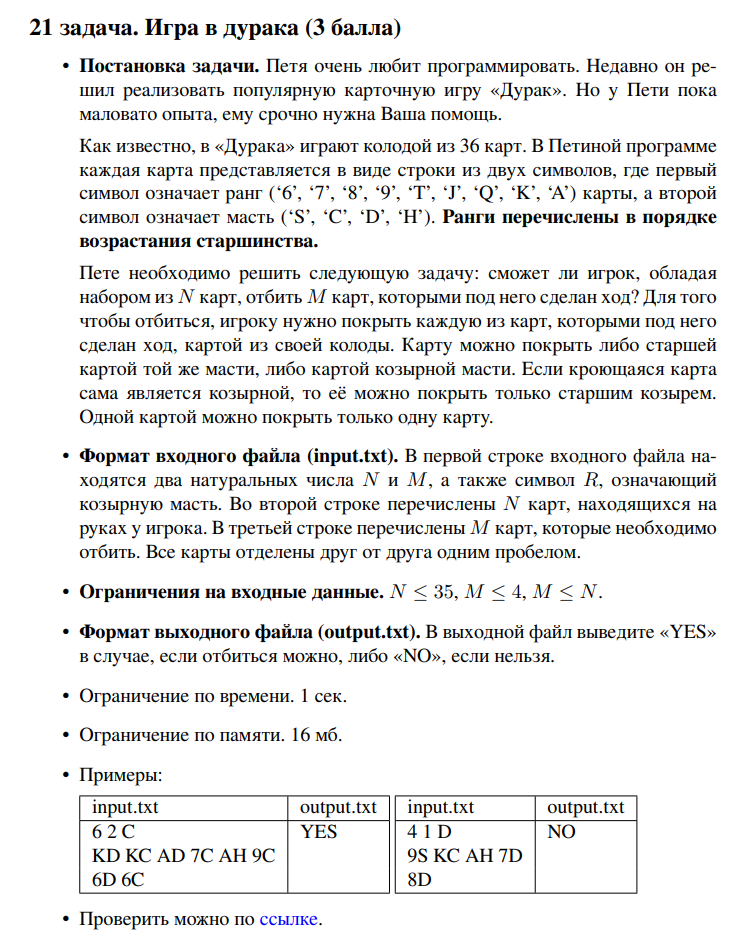




|  | Время выполнения, с |
| --- | --- |
| Пример из задачи | 0.000964 секунд |
| Пример из задачи | 0.001127 секунд |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

## Задача №21. [3 баллов]



**Код программы**

import time

import psutil

import os

def can\_beat\_all\_cards(player\_cards, attack\_cards, trump\_suit):

# Разделение карт игрока на козырные и некозырные

trump\_cards = [card for card in player\_cards if card[1] == trump\_suit]

non\_trump\_cards = [card for card in player\_cards if card[1] != trump\_suit]

# Функция для проверки, может ли карта b побить карту a

def can\_beat(a, b):

if a[1] == b[1]: # Одинаковая масть

return ranks.index(b[0]) > ranks.index(a[0])

if b[1] == trump\_suit: # Карта b козырная

return True

return False

# Сортировка карт по рангу

non\_trump\_cards.sort(key=lambda x: ranks.index(x[0]))

trump\_cards.sort(key=lambda x: ranks.index(x[0]))

for attack\_card in attack\_cards:

beaten = False

for card in non\_trump\_cards:

if can\_beat(attack\_card, card):

non\_trump\_cards.remove(card)

beaten = True

break

if not beaten:

for card in trump\_cards:

if can\_beat(attack\_card, card):

trump\_cards.remove(card)

beaten = True

break

if not beaten:

return "NO"

return "YES"

# Запуск таймера

start\_time = time.time()

# Чтение входных данных из файла

with open('input.txt', 'r') as file:

lines = file.readlines()

# Проверка на наличие хотя бы трех строк в файле

if len(lines) < 3:

raise ValueError("Файл input.txt должен содержать как минимум три строки: количество карт у игрока, количество карт для отбивания и козырную масть, карты игрока и карты для отбивания.")

# Чтение количества карт у игрока, количества карт для отбивания и козырной масти

N, M, R = lines[0].strip().split()

N, M = int(N), int(M)

# Проверка ограничений на N и M

if not (1 <= N <= 35):

raise ValueError("Количество карт у игрока N должно быть в диапазоне от 1 до 35.")

if not (1 <= M <= 4):

raise ValueError("Количество карт для отбивания M должно быть в диапазоне от 1 до 4.")

if not (1 <= M <= N):

raise ValueError("Количество карт для отбивания M не должно превышать количество карт у игрока N.")

# Чтение карт игрока и карт для отбивания

player\_cards = lines[1].strip().split()

attack\_cards = lines[2].strip().split()

# Проверка, что количество карт у игрока соответствует N

if len(player\_cards) != N:

raise ValueError("Количество карт у игрока не соответствует указанному значению N.")

# Проверка, что количество карт для отбивания соответствует M

if len(attack\_cards) != M:

raise ValueError("Количество карт для отбивания не соответствует указанному значению M.")

# Ранги карт в порядке возрастания старшинства

ranks = ['6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A']

# Проверка возможности отбить все карты

result = can\_beat\_all\_cards(player\_cards, attack\_cards, R)

# Запись результата в файл

with open('output.txt', 'w') as file:

file.write(result)

# Остановка таймера и вывод времени выполнения

end\_time = time.time()

execution\_time = end\_time - start\_time

print(f"Время выполнения: {execution\_time:.6f} секунд")

# Проверка использования памяти

process = psutil.Process(os.getpid())

memory\_usage = process.memory\_info().rss / (1024 \* 1024) # Преобразование в мегабайты

print(f"Использование памяти: {memory\_usage:.2f} МБ")

# Проверка ограничения по памяти

if memory\_usage > 16:

raise MemoryError("Использование памяти превысило 16 МБ")

**Текстовое объяснение решения**

После считывания карт в функции can\_beat\_all\_cards, разделяем карты игрока на козырные и некозырные. Определяем вложенную функцию can\_beat, которая проверяет, может ли одна карта побить другую. Сортируем некозырные и козырные карты по рангу. Для каждой атакующей карты проверяем, можно ли её побить сначала некозырными, а затем козырными картами. Если не удается побить карту, возвращаем "NO". Если все карты побиты, возвращаем "YES".

Результат работы кода на примерах из текста задачи:

|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.002059 секунд | 15.23 МБ |
| Пример из задачи | 0.000938999990 секунд | 15.02 МБ |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

# Вывод

В ходе данной лабораторной работы я научился решать задачи. Написанные программы были протестированы, а также были измерены потребляемый ими объём памяти и время работы. Все программы работаю корректно и укладываются в установленные ограничения по времени и памяти на примерах из задач.