

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 №2
Вариант 1

Выполнила:

Гайдук А. С.

К3241

Проверила:

Ромакина О. М.

Санкт-Петербург

2024 г.

Содержание отчета

Содержание отчета.....	2
Задачи по варианту	3
Задача №1. Максимальная стоимость добычи [0.5 балла]	3
Задача №3. Максимальный доход от рекламы [0.5 балла]	5
Задача №8. Расписание лекций [1 балл]	8
Задача №14. Максимальное значение арифметического выражения [2 балла]	11
Задача №17. Ход конём [2.5 балла]	14
Дополнительные задачи	17
Задача №2. Заправки [0.5 балла]	17
Задача №10. Яблоки [1 балл]	20
Задача №11. Максимальное количество золота [1 балл]	23
Задача №13. Сувениры [1.5 балла]	25
Задача №15. Удаление скобок [2 балла]	28
Задача №18. Кафе [2.5 балла]	31
Задача №20. Почти палиндром [3 балла]	35
Задача №21. Игра в дурака [3 балла]	38
Задача №22. Симпатичные узоры [4 балла]	41
Вывод	46

Задачи по варианту

Задача №1. Максимальная стоимость добычи [0.5 балла]

Вор находит гораздо больше добычи, чем может поместиться в его сумке. Помогите ему найти самую ценную комбинацию предметов, предполагая, что любая часть предмета добычи может быть помещена в его сумку. Цель - реализовать алгоритм для задачи о дробном рюкзаке.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def find_max_cost(n, w, items):
    max_cost = 0
    remaining_capacity = w
    items.sort(key=lambda x: x[0] / x[1] if x[1] != 0 else 0,
reverse=True)
    for cost, weight in items:
        if remaining_capacity >= weight:
            max_cost += cost
            remaining_capacity -= weight
        else:
            max_cost += cost * (remaining_capacity / weight)
            break

    return round(max_cost, 4)

with open ('input.txt', 'r') as file:
    n, c = map(int, file.readline().strip().split())
    items = []
    for _ in range(n):
        p, w = map(int, file.readline().split())
        items.append((p, w))

start_time = time.perf_counter()

result = find_max_cost(n, c, items)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open ('output.txt', 'w') as file:
    file.write(f"{result:.4f}\n")

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Для решения данной задачи я создала метод `find_max_cost`, который сортирует добычу по убыванию, деля стоимость предмета на его вес. Затем я прохожусь циклом по всей добыче, если текущая вместимость рюкзака больше веса предмета, то мы добавляем его в рюкзак полностью, добавляя его стоимость к общей стоимости и вычитая его вес из вместимости рюкзака. В другом случае, к максимальной стоимости мы прибавляем стоимость предмета, умноженную на частное общей вместимости и веса предмета, после чего прекращаем обработку остальных предметов, так как рюкзак уже заполнен полностью.

Результат работы кода на примерах из текста задачи:

task1.py		input.txt	output.txt
1	3 50		
2	60 20		
3	100 50		
4	120 30		

task1.py		input.txt	output.txt
1	180.0000		

task1.py		input.txt	output.txt
1	1 10		
2	500 30		

task1.py		input.txt	output.txt
1	166.6667		

Результат работы кода на максимальных и минимальных значениях:

task1.py		input.txt	output.txt
1	1000 2000000		
2	2000000 2000000		
3	2000000 2000000		
4	2000000 2000000		

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000016 сек	0.037653 MB
Пример из задачи	0.000077 сек	0.037707 MB
Пример из задачи	0.000050 сек	0.037660 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000937 сек	0.174148 MB

Вывод по задаче:

Я научилась реализовывать алгоритм дробного рюкзака.

Задача №3. Максимальный доход от рекламы [0.5 балла]

У вас есть n объявлений для размещения на популярной интернет-странице. Для каждого объявления вы знаете, сколько рекламодатель готов платить за один клик по этому объявлению. Вы настроили n слотов на своей странице и оценили ожидаемое количество кликов в день для каждого слота. Теперь ваша цель - распределить рекламу по слотам, чтобы максимизировать общий доход.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def max_ad(a, b):
    a.sort(reverse = True)
    b.sort(reverse=True)
    max_profit = 0
    for i in range(len(a)):
        max_profit += a[i] * b[i]
    return max_profit

with open ('input.txt', 'r') as file:
    n = int(file.readline())
    a = list(map(int, file.readline().strip().split()))
    b = list(map(int, file.readline().strip().split()))

start_time = time.perf_counter()

result = max_ad(a, b)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

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

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Для решения данной задачи я отсортировала списки a, b по убыванию, создала переменную max_profit для хранения промежуточного дохода и прошла циклом по всем элементам списка a, добавив в max_profit произведение i-го элемента a и i-го элемента b.

Результат работы кода на примерах из текста задачи:

task3.py	input.txt	output.txt
1	1	
2	23	
3	39	

task3.py	input.txt	output.txt
1	897	

```
task3.py  input.txt  output.txt
1 3
2 1 3 -5
3 -2 4 1
```

```
task3.py  input.txt  output.txt
1 23
```

Результат работы кода на максимальных и минимальных значениях:

```
task3.py  input.txt  output.txt
1 1
2 1
3 1
```

```
task3.py  input.txt  output.txt
1 1
```

```
task3.py  input.txt  output.txt
1 100000
2 1000000 1000000 1000000 1000000 1000000
   1000000 1000000 1000000 1000000
```

```
task3.py  input.txt  output.txt
1 -1000000000000000000
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000008 сек	0.037456 МВ
Пример из задачи	0.000022 сек	0.037709 МВ
Пример из задачи	0.000013 сек	0.037456 МВ

Верхняя граница диапазона значений входных данных из текста задачи	0.094522 сек	12.942036 MB
--	--------------	--------------

Вывод по задаче:

Я научилась реализовывать алгоритм для нахождения максимальной прибыли от рекламы.

Задача №8. Расписание лекций [1 балл]

Вы наверно знаете, что в ИТМО лекции читают одни из лучших преподавателей мира. К сожалению, лекционных аудиторий у нас не так уж и много, особенно на Биржевой, поэтому каждый преподаватель составил список лекций, которые он хочет прочитать студентам. Чтобы студенты, в начале февраля, увидели расписание лекций, необходимо его составить прямо сейчас. И без вас нам здесь не справиться. У нас есть список заявок от преподавателей на лекции для одной из аудиторий. Каждая заявка представлена в виде временного интервала $[s_i, f_i)$ - время начала и конца лекции. Лекция считается открытым интервалом, то есть какая-то лекция может начаться в момент окончания другой, без перерыва. Необходимо выбрать из этих заявок такое подмножество, чтобы суммарно выполнить максимальное количество заявок. Учтите, что одновременно в лекционной аудитории, конечно же, может читаться лишь одна лекция.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def max_requests(requests):
    max_count = 0
    last_end_time = 0
    requests.sort(key=lambda x: x[1])
    for start, end in requests:
        if start >= last_end_time:
            max_count += 1
            last_end_time = end

    return max_count
```



```

with open ('input.txt', 'r') as file:
    n = int(file.readline())
    requests = []
    for i in range(n):
        start, end = map(int, file.readline().split())
        requests.append((start, end))

start_time = time.perf_counter()

result = max_requests(requests)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

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

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Для решения данной задачи я отсортировала заявки по времени окончания, создала переменные для хранения максимального числа заявок и времени окончания последней лекции. Далее, проходя по времени началу и окончания всех лекций в заявках, я проверяла, что время начала больше времени окончания последней лекции, и если да, то прибавляла к `max_count` единицу, а к времени окончания последней лекции время окончания лекции в текущей заявке.

Результат работы кода на примерах из текста задачи:

The first screenshot shows a code editor with three tabs: 'task8.py', 'input.txt', and 'output.txt'. The 'input.txt' tab is active, showing two lines of input: '1' and '5 10'. The second screenshot shows the same code editor with the 'output.txt' tab active, showing the result '1'.

```
task8.py  input.txt x  output.txt
1      3
2      1 5
3      2 3
4      3 4
```

```
task8.py  input.txt  output.txt x
1      2
```

Результат работы кода на максимальных и минимальных значениях:

```
task8.py  input.txt x  output.txt
1      1000
2      0 1
3      1 2
4      2 3
5      3 4
6      4 5
7      5 6
8      6 7
9      7 8
10     8 9
11     9 10
12    10 11
```

```
task8.py x  input.txt  output.txt x
1      1000
```

```
task8.py  input.txt x  output.txt
1      1
2      1 2
```

```
task8.py  input.txt  output.txt x
1      1
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000016 сек	0.037603 МВ
Пример из задачи	0.000016 сек	0.037605 МВ
Пример из задачи	0.000021 сек	0.037693 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.000399 сек	0.143221 МВ

Вывод по задаче:

Я научилась реализовывать эффективный алгоритм для обработки всех заявок и нахождения максимального подмножества.

Задача №14. Максимальное значение арифметического выражения [2 балла]

В этой задаче ваша цель - добавить скобки к заданному арифметическому выражению, чтобы максимизировать его значение.

$$\max(5 - 8 + 7 \times 4 - 8 + 9) = ?$$

Найдите максимальное значение арифметического выражения, указав порядок применения его арифметических операций с помощью дополнительных скобок.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def max_expr_value(expr):
    n = (len(expr) + 1) // 2

    max_val = [[0] * n for _ in range(n)]
    min_val = [[0] * n for _ in range(n)]

    for i in range(n):
```

```

max_val[i][i] = int(expr[2*i])
min_val[i][i] = int(expr[2 * i])

for length in range(1, n):
    for i in range(n - length):
        j = i + length
        max_val[i][j] = float('-inf')
        min_val[i][j] = float('inf')

        for k in range(i, j):
            a, b = max_val[i][k], max_val[k + 1][j]
            c, d = min_val[i][k], min_val[k + 1][j]
            operation = expr[2 * k + 1]

            if operation == '+':
                max_val[i][j] = max(max_val[i][j], a + b)
                min_val[i][j] = min(min_val[i][j], c + d)
            elif operation == '-':
                max_val[i][j] = max(max_val[i][j], a - d)
                min_val[i][j] = min(min_val[i][j], c - b)
            elif operation == '*':
                max_val[i][j] = max(max_val[i][j], a * b, a * d, c *
b, c * d)
                min_val[i][j] = min(min_val[i][j], a * b, a * d, c *
b, c * d)

        return max_val[0][-1]

with open ('input.txt', 'r') as file:
    expr = file.readline().strip()

start_time = time.perf_counter()

result = max_expr_value(expr)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

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

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Для решения этой задачи я использовала метод динамического программирования, вычисляя максимальные и минимальные значения всех подвыражений через матрицы `max_val` и `min_val`. Сначала я заполнила главную диагональ самими числами из выражения. Далее я начала перебор подвыражений разной длины, для каждого подвыражения `[i, j]` я нахожу максимальные и минимальные значения, инициализировав их изначально с их крайними значениями `(-inf, inf)`. Далее каждое подвыражение пробую разбить на две части через точку разбиения `k`. Здесь `a, b, c, d` – это

максимальные и минимальные значения для левой и правой частей. Так же я выделила оператор, который находится в выражении на позиции $2 * k + 1$. Далее для каждого оператора я рассматриваю все комбинации возможных значений левой и правой частей.

Результат работы кода на примерах из текста задачи:

The figure consists of four vertically stacked screenshots of a Jupyter Notebook interface, illustrating the step-by-step construction of a mathematical expression in a code cell. Each screenshot shows the same notebook with a file browser at the top containing 'task14.py', 'input.txt', and 'output.txt'. The code cell below the browser shows the progression of the expression:

- First screenshot:** The code cell contains the number '1'.
- Second screenshot:** The code cell contains the expression '1+5'.
- Third screenshot:** The code cell contains the expression '1+5-8+7*4-8+9'.
- Fourth screenshot:** The code cell contains the final expression '100'.

Результат работы кода на максимальных и минимальных значениях:

The figure displays four sequential screenshots of a Jupyter Notebook interface, illustrating the execution of a Python script named `task14.py` using different inputs from `input.txt`.

- Top Screenshot:** The notebook shows `task14.py` and `input.txt`. The input in `input.txt` is a long string of 15 '9's. The output in `output.txt` is the number 26.
- Second Screenshot:** The input in `input.txt` is changed to a single '0'. The output in `output.txt` is 0.
- Third Screenshot:** The input in `input.txt` is changed to a single 'p'. The output in `output.txt` is 'p'.
- Bottom Screenshot:** The input in `input.txt` is changed to a single 'p'. The output in `output.txt` is 'p'.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.000063 сек	0.037456 МВ

ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ		
Пример из задачи	0.000088 сек	0.037568 МВ
Пример из задачи	0.000352 секунд	0.037584 МВ
Верхняя граница диапазона значений ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ	0.003552 сек	0.037456 МВ

Вывод по задаче:

Я научилась реализовывать алгоритм, который может находить максимальные значения арифметического выражения с помощью динамического программирования.

Задача №17. Ход конём [2.5 балла]

Шахматная ассоциация решила оснастить всех своих сотрудников такими телефонными номерами, которые бы набирались на кнопочном телефоне ходом коня. Например, ходом коня набирается телефон 340-49-27. При этом телефонный номер не может начинаться ни с цифры 0, ни с цифры 8.

Напишите программу, определяющую количество телефонных номеров длины N, набираемых ходом коня. Поскольку таких номеров может быть очень много, выведите ответ по модулю 10^9 .

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def knight_number(n):
    dp = [[0] * 10 for i in range(n + 1)]

    for digit in range(10):
        if digit != 0 and digit != 8:
            dp[1][digit] = 1

    for length in range(2, n + 1):
        for digit in range(10):
            dp[length][digit] = sum(dp[length - 1][prev] for prev in
moves[digit]) % MOD

    result = sum(dp[n][digit] for digit in range(10)) % MOD
    return result
```

```

MOD = 10**9

moves = {
    0: [4, 6],
    1: [6, 8],
    2: [7, 9],
    3: [4, 8],
    4: [0, 3, 9],
    5: [],
    6: [0, 1, 7],
    7: [2, 6],
    8: [1, 3],
    9: [2, 4]
}

with open ('input.txt', 'r') as file:
    n = int(file.readline())

start_time = time.perf_counter()

result = knight_number(n)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

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

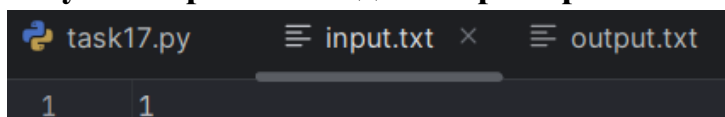
print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

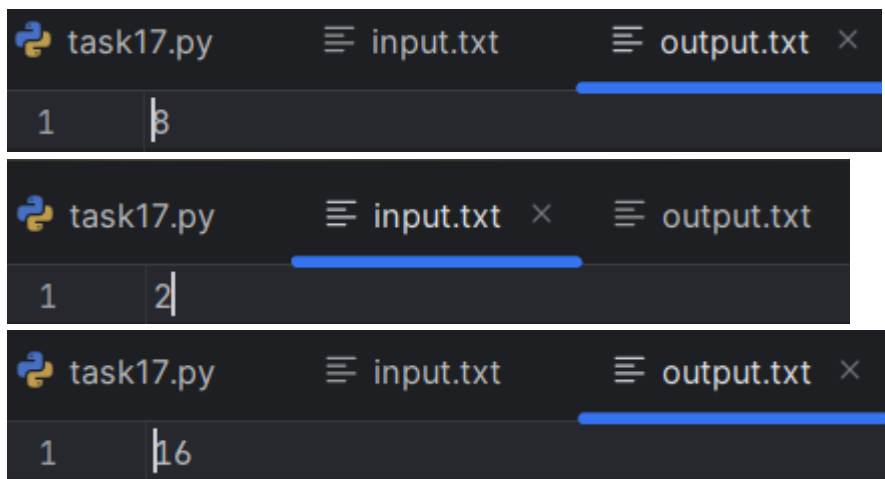
```

Текстовое объяснение решения:

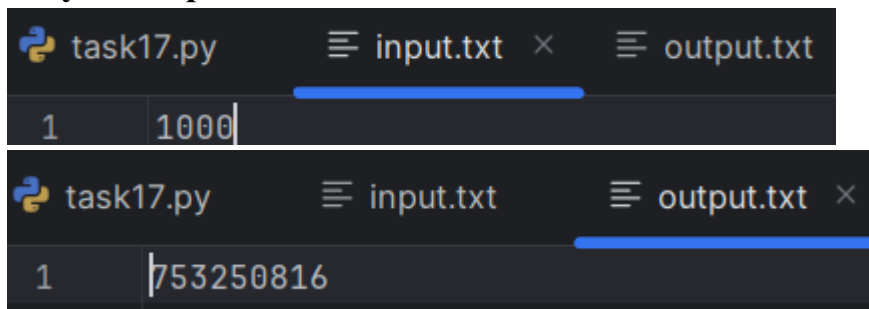
Я создала таблицу размером $(n+1)*10$ (количество длин на количество цифр) для хранения количества способов набрать номера, а также обозначила возможные ходы «конем». Для номера длины 1 я задала начальные значения, исключив 0 и 8. Далее цикл идет по всем длинам от 2 до n . Здесь чтобы набрать номер длины $length$, заканчивающийся на цифру $digit$, нужно добавить к номеру длины $length - 1$ цифру $digit$, если возможен переход в $digit$ через $moves$. Суммируем все подходящие номера и берем результат по модулю 10^9 , чтобы избежать переполнения. После завершения заполнения таблицы dp для всех длин, суммируем все значения $dp[n][digit]$ и снова берем результат по модулю 10^9 .

Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000040 сек	0.038106 МВ
Пример из задачи	0.000080 сек	0.038106 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.045987 сек	0.452447 МВ

Вывод по задаче:

В данной задаче я реализовала алгоритм поиска количества номеров определенной длины, которые набираются ходом коня.

Дополнительные задачи

Задача №2. Заправки [0.5 балла]

Вы собираетесь поехать в другой город, расположенный в d км от вашего родного города. Ваш автомобиль может проехать не более m км на полном баке, и вы начинаете с полным баком. По пути есть заправочные станции на расстояниях $stop_1, stop_2, \dots, stop_n$ из вашего родного города. Какое минимальное количество заправок необходимо?

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def min_refills(d, m, stops):
    stops = [0] + stops + [d]
    num_refills = 0
    current_refill = 0

    while current_refill < len(stops) - 1:
        last_refill = current_refill

        while current_refill < len(stops) - 1 and (stops[current_refill + 1] - stops[last_refill] <= m):
            current_refill += 1
        if current_refill == last_refill:
            return -1
        if current_refill < len(stops) - 1:
            num_refills += 1

    return num_refills

with open('input.txt', 'r') as file:
    d = int(file.readline())
    m = int(file.readline())
    n = int(file.readline())
    stops = list(map(int, file.readline().split()))

start_time = time.perf_counter()

result = min_refills(d, m, stops)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

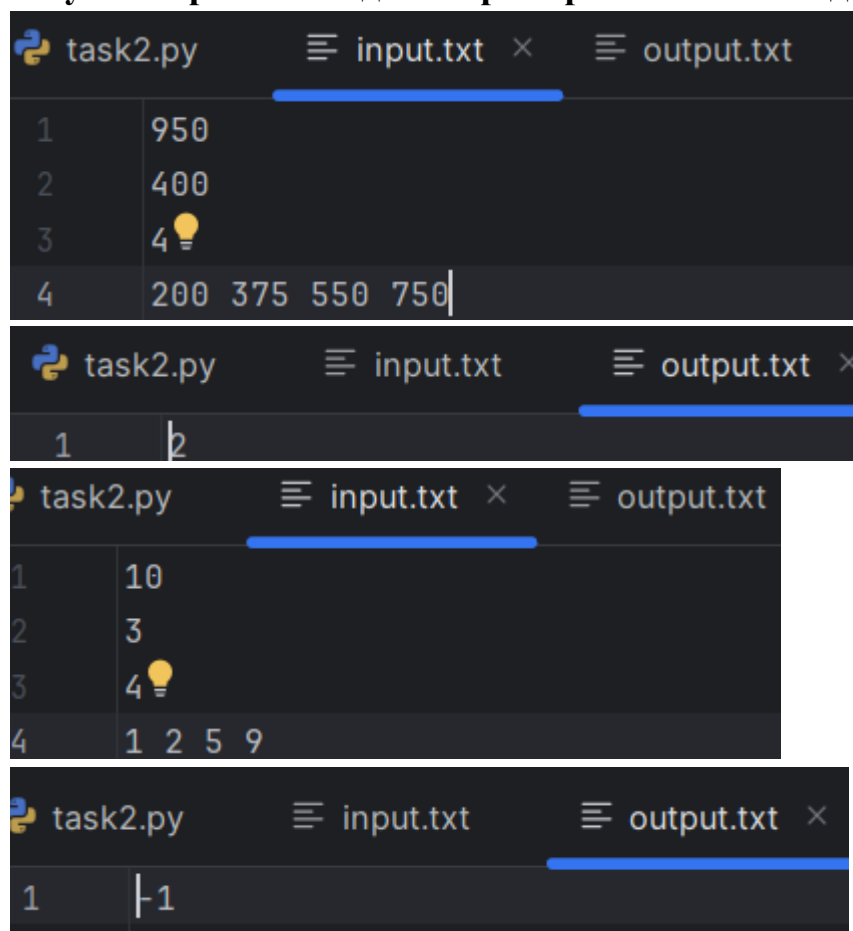
with open('output.txt', 'w') as file:
    file.write(str(result))

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Я добавила начальную и конечную точку в список остановок, чтобы отобразить полную информацию о маршруте. Далее я ввела счетчик заправок и индекс текущей позиции, на которой находится машина. Пока машина не достигнет конечной станции, мы сохраняем текущую станцию в `last_refill`. Внутренний цикл перемещает текущую позицию к следующей доступной, до которой можно доехать с текущим запасом топлива. Если машина осталась на той же станции, то следующая станция слишком далеко, и путь невозможен. Если мы еще не достигли конечной точки, то увеличиваем счетчик заправок.

Результат работы кода на примерах из текста задачи:



The image shows four screenshots of a code editor with the file `task2.py` open. Each screenshot shows the input and output of the program for a specific test case.

- First screenshot:** The `input.txt` tab is active. It contains four lines of input: `950`, `400`, `4` (with a lightbulb icon), and `200 375 550 750`. The `output.txt` tab is also visible but empty.
- Second screenshot:** The `output.txt` tab is active. It contains the output `2` for the first test case.
- Third screenshot:** The `input.txt` tab is active. It contains four lines of input: `10`, `3`, `4` (with a lightbulb icon), and `1 2 5 9`. The `output.txt` tab is also visible but empty.
- Fourth screenshot:** The `output.txt` tab is active. It contains the output `1` for the second test case.

```
task2.py  input.txt × output.txt
200
250
2
100
150
```

```
task2.py  input.txt  output.txt ×
1  |
```

Результат работы кода на максимальных и минимальных значениях:

```
task2.py ×  input.txt ×  output.txt
1  100000
2  400
3  300
4  400 800 1200 1600 2000 2400 2800 3200
```

```
task2.py  input.txt  output.txt ×
1  299
```

```
task2.py  input.txt ×  output.txt
1
1
1
1
```

```
task2.py  input.txt  output.txt ×
1  0
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.000026 сек	0.037531 МВ

ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ		
Пример из задачи	0.000036 сек	0.037661 МВ
Пример из задачи	0.000034 сек	0.037586 МВ
Пример из задачи	0.000017 сек	0.037456 МВ
Верхняя граница диапазона значений ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ	0.001941 сек	0.060118 МВ

Вывод по задаче:

Я научилась реализовывать алгоритм подсчета количества заправок для достижения назначенной точки, используя жадные алгоритмы.

Задача №10. Яблоки [1 балл]

Алисе в стране чудес попались n волшебных яблок. Про каждое яблоко известно, что после того, как его съешь, твой рост сначала уменьшится на a_i сантиметров, а потом увеличится на b_i сантиметров. Алиса очень голодная и хочет съесть все n яблок, но боится, что в какой-то момент ее рост s станет равным нулю или еще меньше, и она пропадет совсем. Помогите ей узнать, можно ли съесть яблоки в таком порядке, чтобы в любой момент времени рост Алисы был больше нуля.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def apples_count(n, s, apples):
    eaten_apples = []
    for i in range(n):
        for j in range(n):
            if s > 0 and apples[j][0] <= s:
                eaten_apples.append(j + 1)
                s -= apples[j][0]
                s += apples[j][1]
                apples[j] = (float('inf'), float('inf'))
                break
        else:
            return -1
    return eaten_apples
```

```

with open ('input.txt', 'r') as file:
    n, s = map(int, file.readline().split())
    apples = [tuple(map(int, line.split())) for line in file]

start_time = time.perf_counter()

result = apples_count(n, s, apples)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open ('output.txt', 'w') as file:
    if result == -1:
        file.write(str(result))
    else:
        file.write(' '.join(map(str, result)))

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Я создала пустой список для добавления индексов съеденных яблок. Далее я прохожусь по всем яблокам и проверяю, что рост Алисы положителен и что текущее яблоко можно съесть (что рост не станет 0 или меньше), далее добавляю съеденное яблоко в список и обновляю рост Алисы. Чтобы яблоко больше не рассматривалось, заменяю его значения на недостижимые. Если не осталось яблок, которые можно съесть, возвращаю -1.

Результат работы кода на примерах из текста задачи:

task10.py	input.txt	output.txt
1	3 5	
2	2 3	
3	10 5	
4	5 10	

task10.py	input.txt	output.txt
1	1 3 2	

```

task10.py  input.txt  output.txt
1 3 5
2 2 3
3 10 5
4 5 6

task10.py  input.txt  output.txt
1 1

```

Результат работы кода на максимальных и минимальных значениях:

```

task10.py  input.txt  output.txt
1 1 1
2 1 1

task10.py  input.txt  output.txt
1 1

task10.py  input.txt  output.txt
1 1000 1000
2 1000 1000
3 1000 1000

task10.py  input.txt  output.txt
1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000031 сек	0.037636 МВ
Пример из задачи	0.000032 сек	0.037824 МВ
Пример из задачи	0.000034 сек	0.037822 МВ

Верхняя граница диапазона значений входных данных из текста задачи	0.102775 сек	0.166636 МВ
--	--------------	-------------

Вывод по задаче:

В данной задаче я реализовала алгоритм вывода номеров яблок, которые может съесть Алиса, используя жадные алгоритмы.

Задача №11. Максимальное количество золота [1 балл]

Текст задачи.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def max_gold(w, weights):
    dp = [0] * (w + 1)

    for weight in weights:
        for j in range(w, weight - 1, -1):
            dp[j] = max(dp[j], dp[j - weight] + weight)

    return dp[w]

with open('input.txt', 'r') as file:
    w, n = map(int, file.readline().split())
    weights = list(map(int, file.readline().split()))

start_time = time.perf_counter()

result = max_gold(w, weights)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

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

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое  
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Я создала массив dp, который хранит максимальный вес рюкзака. Перебирая каждый слиток из списка, для каждой вместимости j, если

текущий слиток может быть добавлен, то $dp[j]$ обновляется как максимум между текущим значением и суммой веса слитка и $dp[j - \text{вес слитка}]$.

Результат работы кода на примерах из текста задачи:

```
task11.py  input.txt  output.txt
1  10 3
2  1 4 8
```

```
task11.py  input.txt  output.txt
1  9
```

Результат работы кода на максимальных и минимальных значениях:

```
task11.py  input.txt  output.txt
1  10000 300
2  100000 100000 100000 100000 100000
```

```
task11.py  input.txt  output.txt
1  0
```

```
task11.py  input.txt  output.txt
1  1 1
2  0
```

```
task11.py  input.txt  output.txt
1  0
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000014 сек	0.037575 МВ
Пример из задачи	0.000028 сек	0.037626 МВ

Верхняя граница диапазона значений входных данных из текста задачи	0.000552 сек	0.116115 МВ
--	--------------	-------------

Вывод по задаче:

Я реализовала алгоритм, который считает, какой вес в слитках может поместиться в рюкзак с помощью динамического программирования.

Задача №13. Сувениры [1.5 балла]

Вы и двое ваших друзей только что вернулись домой после посещения разных стран. Теперь вы хотели бы поровну разделить все сувениры, которые все трое накопили.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def souvenirs(v):
    total_sum = sum(v)
    if total_sum % 3 != 0:
        return 0

    target_sum = total_sum // 3
    n = len(v)

    dp = [0] * (target_sum + 1)
    dp[0] = 1

    for i in v:
        for j in range(target_sum, i - 1, -1):
            if dp[j - i] > 0:
                dp[j] += 1

    return 1 if dp[target_sum] >= 3 else 0

with open('input.txt', 'r') as file:
    n = file.readline()
    v = list(map(int, file.readline().split()))

start_time = time.perf_counter()

result = souvenirs(v)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open('output.txt', 'w') as file:
    file.write(str(result))
```

```
print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Сначала я считала общую сумму всех сувениров, и проверила, что сумма делится на три (иначе возвращаем 0). Далее я выделила целевую сумму для каждого из трех подмножеств и создала массив, который показывает, сколько раз можно достигнуть определенную сумму. Для каждого сувенира обновляем DP с конца: если можно достичь сумму $(j - 1)$, добавляем сувенир I , чтобы получить новую сумму j . Если целевую сумму можно достичь 3 и более раз, возвращаем единицу, иначе 0.

Результат работы кода на примерах из текста задачи:

```
task13.py  input.txt  output.txt
1  11
2  17 59 34 57 17 23 67 1 18 2 59
```

```
task13.py  input.txt  output.txt
1  1
```

```
task13.py  input.txt  output.txt
1  4
2  3 3 3 3
```

```
task13.py  input.txt  output.txt
1  1
```

```
task13.py  input.txt  output.txt
1  1
2  40
```

```
task13.py  input.txt  output.txt
1  0
```

Результат работы кода на максимальных и минимальных значениях:

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000015 сек	0.037456 МВ
Пример из задачи	0.000113 сек	0.037670 МВ
Пример из задачи	0.000027 сек	0.037622 МВ
Пример из задачи	0.000009 сек	0.037612 МВ
Пример из задачи	0.000057 сек	0.037668 МВ
Верхняя граница диапазона значений	0.000184 сек	0.037456 МВ

ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ		
------------------------------------	--	--

Вывод по задаче:

Я научилась реализовывать алгоритм деления сувениров на три, исходя из суммы каждого предмета. В этом мне помогли методы динамического программирования.

Задача №15. Удаление скобок [2 балла]

Дана строка, составленная из круглых, квадратных и фигурных скобок. Определите, какое наименьшее количество символов необходимо удалить из этой строки, чтобы оставшиеся символы образовывали правильную скобочную последовательность.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def bracket_seq(s):
    n = len(s)

    if n < 2:
        return ""

    dp = [[0] * n for _ in range(n)]
    matching = {'(': ')', '[': ']', '{': '}'

    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] in "([{":
                if s[j] in ")]}" and matching[s[j]] == s[i]:
                    dp[i][j] = dp[i + 1][j - 1] + 2

            for k in range(i, j):
                dp[i][j] = max(dp[i][j], dp[i][k] + dp[k + 1][j])

    i, j = 0, n - 1
    result = []
    while i <= j:
        if s[i] in "([{":
            if s[j] in ")]}" and matching.get(s[j]) == s[i]:
                result.append(s[i])
                result.append(s[j])
                i += 1
                j -= 1
            else:
                if dp[i + 1][j] >= dp[i][j - 1]:
```

```

        i += 1
    else:
        j -= 1
    else:
        i += 1

    return ''.join(result)

with open ('input.txt', 'r') as file:
    s = file.readline().strip()

start_time = time.perf_counter()

results = bracket_seq(s)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

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

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Я создала список, который хранит максимальную длину правильной скобочной последовательности в подстроке от индекса i до j , а также словарь `matching`, который сопоставляет закрывающую скобку ее открывающей для проверки корректности пар. Далее я заполняю таблицу `dp`, проходя по всем подстрокам длины от 2 до n , для каждой подстроки от индекса i до j , если крайние символы образуют корректную пару, то длина ПСП увеличивается на 2. Для всех разбиений подстроки на 2 части через точку разбиения k , проверяю, что длина ПСП может быть увеличена, если объединить результаты двух частей.

После заполнения таблицы, начинаю восстанавливать результат с первой и последней позиции строки. Если открывающая и закрывающая скобка образуют пару, добавляем их в результат, сами смещаем индексы внутрь. Иначе, сравнением значения результатов без крайней левой и крайней правой скобки и убираем символ, который не участвует в максимальной ПСП.

Результат работы кода на примерах из текста задачи:

Вывод по задаче:

Я реализовала алгоритм, который способен образовать правильную скобочную последовательность максимальной длины через удаление скобок. В этом мне помогли методы динамического программирования.

Задача №18. Кафе [2.5 балла]

Около университета недавно открылось новое кафе, в котором действует следующая система скидок: при каждой покупке более чем на 100 рублей покупатель получает купон, дающий право на один бесплатный обед (при покупке на сумму 100 рублей и меньше такой купон покупатель не получает). Однажды вам на глаза попался прейскурант на ближайшие n дней. Внимательно его изучив, вы решили, что будете обедать в этом кафе все n дней, причем каждый день вы будете покупать в кафе ровно один обед. Однако стипендия у вас небольшая, и поэтому вы хотите по максимуму использовать предоставляемую систему скидок так, чтобы ваши суммарные затраты были минимальны. Требуется найти минимально возможную суммарную стоимость обедов и номера дней, в которые вам следует воспользоваться купонами.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def find_min_cost(n, costs):
    costs = [0] + costs
    dp = [[float('inf')] * (n+1) for _ in range(n+1)]
    dp[0][0] = 0

    for i in range(1, n+1):
        for j in range(0, n):
            if costs[i] <= 100:
                dp[i][j] = min(dp[i-1][j] + costs[i], dp[i-1][j+1])
            else:
                dp[i][j] = min(dp[i-1][j-1] + costs[i], dp[i-1][j+1])
    min_cost = min(dp[n])
    unused_cupons = dp[n].index(min_cost)

    cur_min = min_cost
    days = []
    for i in range(n, 0, -1):
        if cur_min in dp[i-1]:
            days.append(i)
        else:
            cur_min -= costs[i]
```

```

        used_cupons = len(days)

        return min_cost, unused_cupons, used_cupons, days[::-1]

with open ('input.txt', 'r') as file:
    n = int(file.readline())
    costs = list(map(int, file.readlines()))

start_time = time.perf_counter()

min_cost, unused_cupons, used_cupons, day_coupon = find_min_cost(n,
costs)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open ('output.txt', 'w') as file:
    file.write(str(min_cost) + '\n')
    file.write(f"{unused_cupons} {used_cupons}\n")
    file.write(" ".join(map(str, day_coupon)) + "\n")

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Я добавила 0 в начало массива costs, чтобы использовать индексацию с 1-го дня. Далее я создала таблицу ДП, где $dp[i][j]$ – это минимальная стоимость обедов за i дней, если есть j неистраченных купонов. Установим $dp[0][0] = 0$, все остальные равны плюс бесконечности.

Далее проходимся по всем дням и находим минимальную стоимость для всех вариантов купонов. Если стоимость текущего обеда больше 100, то значение находим как минимум между стоимостью блюд количеством $i-1$ и с купонами $j-1$ (так как за обед мы получим купон), если мы покупаем обед, и блюд количеством $i-1$ и с купонами $j+1$ + стоимость блюда, если мы тратим купон.

Если стоимость текущего блюда меньше или равна 100, то находим значение в ячейке как минимум из блюд количеством $i-1$ и с купонами j , если мы покупаем обед, и блюд количеством $i-1$ и с купонами $j+1$, если тратим купон.

Восстанавливаем дни, в которые использовались купоны. Проходясь по дням в обратном порядке, мы смотрим на минимальную сумму в строке выше, если там она присутствует, то был использован купон, иначе обед купили и мы вычитаем его стоимость. Продолжаем сравнение с новой минимальной суммой.

Ответ – это $\min(dp[n])$, количество неиспользованных купонов – это индекс, на котором хранится ответ, количество использованных купонов – количество дней, в которые был применен купон, дни записаны в массиве при восстановлении ответа.

Результат работы кода на примерах из текста задачи:

```
task18.py  input.txt  output.txt
1 5
2 110
3 40
4 120
5 110
6 60
```

```
task18.py  input.txt  output.txt
1 260
2 0 2
3 3 5
```

```
task18.py  input.txt  output.txt
3
110
110
110
```

```
task18.py  input.txt  output.txt
1 220
2 1 1
3 3
```

Результат работы кода на максимальных и минимальных значениях:

```
Project 3.py  input.txt  output.txt
1 100
2 300
3 300
4 300
5 300
6 300
```

```
task18.py  input.txt  output.txt
1 15000
2 0 50
3 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65
```

```
task18.py  input.txt  output.txt
1 0
2 0
```

```
task18.py  input.txt  output.txt
1 0
2 0 0
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000031 сек	0.037589 MB
Пример из задачи	0.000077 сек	0.037831 MB
Пример из задачи	0.000069 сек	0.037735 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.015660 сек	0.375059 MB

Вывод по задаче:

Я реализовала алгоритм подсчета минимальной возможной суммарной стоимости обедов используя динамическое программирование.

Задача №20. Почти палиндром [3 балла]

Слово называется палиндромом, если его первая буква совпадает с последней, вторая – с предпоследней и т. д. Например: «abba», «madam», «x».

Для заданного числа K слово называется почти палиндромом, если в нем можно изменить не более K любых букв так, чтобы получился палиндром. Например, при $K = 2$ слова «reactor», «kolobok», «madam» являются почти палиндромами (подчеркнуты буквы, заменой которых можно получить палиндром).

Подсловом данного слова являются все слова, получающиеся путем вычеркивания из данного нескольких (возможно, одной или нуля) первых букв и нескольких последних. Например, подсловами слова «cat» являются слова «с», «а», «т», «са», «ат» и само слово «cat» (а «ct» подсловом слова «cat» не является).

Требуется для данного числа K определить, сколько подслов данного слова S являются почти палиндромами.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def count_almost_palindromes(s, n, k):
    dp = [[0] * n for _ in range(n)]
    count = 0

    for length in range(1, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if i == j:
                dp[i][j] = 0
            elif s[i] == s[j]:
                dp[i][j] = dp[i + 1][j - 1] if i + 1 <= j - 1 else 0
            else:
                dp[i][j] = (dp[i + 1][j - 1] if i + 1 <= j - 1 else 0) + 1

            if dp[i][j] <= k:
                count += 1
```

```

return count

with open ('input.txt', 'r') as file:
    n, k = map(int, file.readline().strip().split())
    s = file.readline()

start_time = time.perf_counter()

result = count_almost_palindromes(s, n, k)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

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

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Я создала таблицу `dp` для хранения минимального количества изменений символов, необходимых для превращения слова в палиндром. Перебирая длины подстрок от 1 до `n` и начальные позиции `i` подстроки в строке (`i` меняется от 0 до `n - length`, чтобы подстрока не выходила за пределы строки), вычисляю конечную позицию `j` на основе начальной. Если подстрока состоит из одного символа, замен не требуется. Если крайние символы подстроки совпадают, тогда количество замен для подстроки равно количеству замен для внутренней подстроки (для этого сначала проверяем, существует ли подстрока между `i`, `j`). Если же крайние символы не совпадают, нужно заменить один из символов, чтобы они стали одинаковыми (значит прибавляем к `dp[i][j]` единицу) и определяем значение внутренней подстроки (если она есть). Если количество замен для подстроки меньше или равно `k`, то эта подстрока – почти палиндром, увеличиваем счетчик на единицу.

Результат работы кода на примерах из текста задачи:

task20.py	input.txt	output.txt
1	3 3	
2	aaa	

```
task20.py × input.txt output.txt ×
1 |b

task20.py input.txt × output.txt
1 | 5 1
2 | abcde|

task20.py input.txt output.txt ×
1 |b2
```

Результат работы кода на максимальных и минимальных значениях:

```
task20.py input.txt × output.txt
1 | 5000 5000
2 | aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

task20.py input.txt output.txt ×
1 |b2502500

task20.py input.txt × output.txt
1 | 1 0
2 | c|

task20.py input.txt output.txt ×
1 | 1
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000021 сек	0.037575 МВ
Пример из задачи	0.000031 сек	0.037621 МВ
Пример из задачи	0.000021 сек	0.037625 МВ

Верхняя граница диапазона значений входных данных из текста задачи	1.318048 сек	200.357327 MB
--	--------------	---------------

Вывод по задаче:

Я реализовала алгоритм для подсчета количества возможных «почти палиндромов» при использовании динамического программирования.

Задача №21. Игра в дурака [3 балла]

Петя очень любит программировать. Недавно он решил реализовать популярную карточную игру «Дурак». Но у Пети пока маловато опыта, ему срочно нужна Ваша помощь.

Как известно, в «Дурака» играют колодой из 36 карт. В Петиной программе каждая карта представляется в виде строки из двух символов, где первый символ означает ранг ('6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A') карты, а второй символ означает масть ('S', 'C', 'D', 'H'). Ранги перечислены в порядке возрастания старшинства.

Пете необходимо решить следующую задачу: сможет ли игрок, обладая набором из N карт, отбить M карт, которыми под него сделан ход? Для того чтобы отбиться, игроку нужно покрыть каждую из карт, которыми под него сделан ход, картой из своей колоды. Карту можно покрыть либо старшей картой той же масти, либо картой козырной масти. Если кроющаяся карта сама является козырной, то её можно покрыть только старшим козырем. Одной картой можно покрыть только одну карту.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

suits = 'SCDH'
ranks = '6789TJQKA'

def cards_matrix(player_cards):
    dp = [[False] * 4 for _ in range(9)]
    for card in player_cards:
        rank_idx = ranks.index(card[0])
        suit_idx = suits.index(card[1])
        dp[rank_idx][suit_idx] = True
```

```

return dp

def can_defend(n, r, player_cards, enemy_cards):

    trump_suit_idx = suits.index(r)
    dp = cards_matrix(player_cards)
    for card in enemy_cards:
        enemy_rank, enemy_suit = ranks.index(card[0]),
suits.index(card[1])
        card_covered = False

        for rank in range(enemy_rank + 1, 9):
            if dp[rank][enemy_suit]:
                dp[rank][enemy_suit] = False
                card_covered = True
                break

        if not card_covered and enemy_suit != trump_suit_idx:
            for rank in range(9):
                if dp[rank][trump_suit_idx]:
                    dp[rank][trump_suit_idx] = False
                    card_covered = True
                    break

        if not card_covered:
            return 'NO'

    return 'YES'

with open ('input.txt', 'r') as file:
    n, m, r = file.readline().strip().split()
    player_cards = list(file.readline().strip().split())
    enemy_cards = list(file.readline().strip().split())

start_time = time.perf_counter()

result = can_defend(n, r, player_cards, enemy_cards)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

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

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Я объявила строки для мастей и рангов карт, а также создала два метода. Первый метод создает матрицу 9*4, представляющую наличие карт у игрока. Каждый элемент матрицы изначально False (нет карты такого ранга и масти). Для каждой карты игрока находим индексы ранга и масти и отмечаем, что карта с этим рангом и мастью присутствует в колоде (флаг

True). Второй метод проверяет, может ли игрок отбиться. Сначала я нахожу индекс козырной масти и создаю матрицу наличия карт игрока через первый метод. Прохожусь циклом по атакующим картам и для каждой карты нахожу индексы ранга и масти и устанавливаю флаг `card_covered` в `False` (карта пока не покрыта). Далее проверяю все карты игрока той же масти, начиная с карт старшего ранга, если такую нахожу – убираю её из колоды и устанавливаю флаг `card_covered = True`. Если карту не удалось покрыть картой той же масти, проверяю козыри (если атакующая карта не козырная). Ищу любую козырную карту и, если нахожу подходящую, убираю её из колоды и устанавливаю флаг `card_covered = True`. Если не удалось покрыть карту, возвращаю `NO`, иначе – `YES`.

Результат работы кода на примерах из текста задачи:

```
task21.py  input.txt  output.txt
1 6 2 C
2 KD KC AD 7C AH 9C
3 6D 6C
```

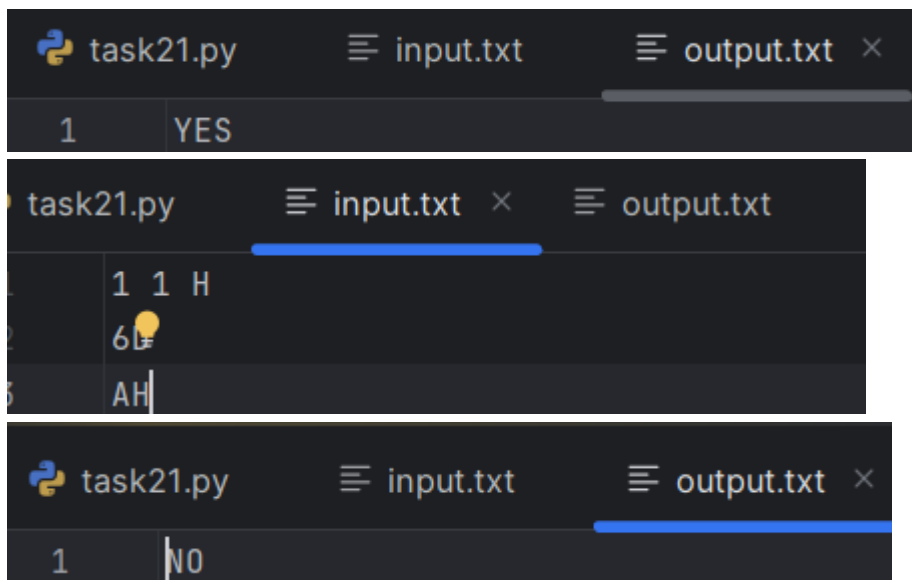
```
task21.py  input.txt  output.txt
1 YES
```

```
task21.py  input.txt  output.txt
1 4 1 D
2 9C KC AH 7D
3 8D
```

```
task21.py  input.txt  output.txt
1 NO
```

Результат работы кода на максимальных и минимальных значениях:

```
task21.py  input.txt  output.txt
1 35 4 H
2 6C 6D 6H 6S 7C 7D 7H 7S 8C 8D 8H 8S 9C 9D
3 AS
```

Проверка на астр:

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
22721084	10.01.2025 3:52:39	Гайдук Алина Сергеевна	0698	Python	Accepted		0,078	1434 Кб

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000030 сек	0.037616 MB
Пример из задачи	0.000034 сек	0.038120 MB
Пример из задачи	0.000030 сек	0.037616 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000035 сек	0.037616 MB

Вывод по задаче:

С помощью методов динамического программирования мне удалось реализовать алгоритм, выполняющий функционал игры в «Дурака».

Задача №22. Симпатичные узоры [4 балла]

Компания BrokenTiles планирует заняться выкладыванием во дворах у состоятельных клиентов узор из черных и белых плиток, каждая из которых

имеет размер 1×1 метр. Известно, что дворы всех состоятельных людей имеют наиболее модную на сегодня форму прямоугольника $M \times N$ метров. Однако при составлении финансового плана у директора этой организации появилось целых две серьезных проблемы: во-первых, каждый новый клиент очевидно захочет, чтобы узор, выложенный у него во дворе, отличался от узоров всех остальных клиентов этой фирмы, а во вторых, этот узор должен быть симпатичным. Как показало исследование, узор является симпатичным, если в нем нигде не встречается квадрата 2×2 метра, полностью покрытого плитками одного цвета.

Для составления финансового плана директору необходимо узнать, сколько клиентов он сможет обслужить, прежде чем симпатичные узоры данного размера закончатся. Помогите ему!

Листинг кода:

```
from itertools import product
from functools import reduce
import tracemalloc
import time

tracemalloc.start()

def generate_patterns(n):
    return list(product([False, True], repeat=n))

def is_compatible(a, b):
    for i in range(len(a) - 1):
        if a[i] == b[i] == a[i + 1] == b[i + 1]:
            return False
    return True

def build_compatibility_matrix(patterns):
    size = len(patterns)
    matrix = [[False] * size for _ in range(size)]
    for i in range(size):
        for j in range(i, size):
            compatible = is_compatible(patterns[i], patterns[j])
            matrix[i][j] = matrix[j][i] = compatible
    return matrix

def count_pretty_patterns(matrix, m):
    size = len(matrix)
    dp = [[0] * size for _ in range(m)]
    for i in range(size):
        dp[0][i] = 1

    for i in range(1, m):
        for j in range(size):
            for k in range(size):
                if matrix[k][j]:
                    dp[i][j] += dp[i - 1][k]

    return dp
```

```

with open('input.txt', 'r') as file:
    n, m = map(int, file.readline().strip().split())

start_time = time.perf_counter()

max_dim = max(n, m)
min_dim = min(n, m)

patterns = generate_patterns(min_dim)

compatibility_matrix = build_compatibility_matrix(patterns)

dp_table = count_pretty_patterns(compatibility_matrix, max_dim)

result = sum(dp_table[-1])

end_time = time.perf_counter()

with open('output.txt', 'w') as file:
    file.write(str(result) + '\n')

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Я создала четыре метода. Первый метод создает все возможные строки длины n из плиток двух цветов (`false` – черная плитка, `true` – белая плитка), используя `itertools.product`, который берет элементы и создает всевозможные комбинации. Второй метод проверяет, могут ли две строки узора быть одна под другой, чтобы не образовывать запрещенные квадраты 2×2 из плиток одного цвета. Для этого я прохожу по всем парам соседних плиток строк a , b и проверяю для каждого i , образуется ли квадрат. Третий метод строит матрицы совместимости: сначала создаю пустую матрицу размером $size \times size$, для каждой пары строк вызывается второй метод (проверка совместимости), если строки совместимы, записываем в `matrix[i][j]` и `matrix[j][i]` (из-за симметрии) `True`. Четвертый метод использует матрицу совместимости для подсчета всех симпатичных узоров высоты m . Сначала я создаю таблицу `dp`, где `dp[i][j]` – количество симпатичных узоров высоты $i + 1$, оканчивающихся строкой j . На высоте 1 любая строка может быть начальной, поэтому `dp[0][j] = 1` для всех j . Для

каждой высоты i и строки j суммирую количество узоров предыдущей высоты $i - 1$, заканчивающихся строками k , которые совместимы с j (проверяю через матрицу).

Результат работы кода на примерах из текста задачи:

task22.py × input.txt × output.txt

1 2 2

task22.py input.txt output.txt ×

1 1 4

task22.py input.txt × output.txt

1 3 3

task22.py input.txt output.txt ×

1 3 2 2

Результат работы кода на максимальных и минимальных значениях:

task22.py input.txt × output.txt

1 5 6

task22.py input.txt output.txt ×

1 119310334

task22.py input.txt × output.txt

1 1 1

task22.py input.txt output.txt ×

1 2

Проверка на астр:

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
22721158	10.01.2025 5:03:43	Гайдук Алина Сергеевна	0083	Python	Accepted		0,078	1326 Кб

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000075 сек	0.038048 МВ
Пример из задачи	0.000080 сек	0.038048 МВ
Пример из задачи	0.000267 сек	0.038048 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.002962 сек	0.040499 МВ

Вывод по задаче:

С помощью методов динамического программирования мне удалось реализовать алгоритм, который эффективно и быстро считает количество возможных симпатичных узоров.

Вывод

В ходе выполнения данной лабораторной работы я потренировалась в составлении алгоритмов с использованием методов динамического программирования. Также я прорешала задачи на жадные алгоритмы.