Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
ПО КУРСУ «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»
ТЕМА: ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ
ВАРИАНТ 10

Выполнил:
Соболев А. А
K3240

Проверила:
Ромакина О. М.

Санкт-Петербург
2025

СОДЕРЖАНИЕ

Задачи по варианту 3
3 задача. Максимальный доход от рекламы (0.5 балла) 3
7 задача. Проблема сапожника (0.5 балла) 5
9 задача. Распечатка (1 балл) 7
15 задача. Удаление скобок (2 балла) 10
21 задача. Игра в дурака (3 балла) 12
Дополнительные задачи 15
1 задача. Максимальная стоимость добычи (0.5 балла) 15
2 задача. Заправки (0.5 балла) 17
8 задача. Расписание лекций (1 балл) 19
17 задача. Ход конём (3 балла) 21
18 задача. Кафе (2.5 балла) 23
20 задача. Почти палиндром (3 балла) 26
Вывод 28

ЗАДАЧИ ПО ВАРИАНТУ

3 задача. Максимальный доход от рекламы (0.5 балла)

Постановка задачи

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.

Ваша цель - распределить рекламу по слотам, чтобы максимизировать общий доход.

Листинг кода

```
import time
import tracemalloc

def find_max(a, b):
    a.sort()
    b.sort()

    return sum(a[i] * b[i] for i in range(len(a)))

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

start_time = time.perf_counter()
tracemalloc.start()

result = find_max(a, b)

end_time = time.perf_counter()

current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")
with open("output.txt", "w") as file:
    file.write(st(result) + "\n")
```

Объяснение кода

В данной задаче я отсортировал по возрастанию оба массива и перемножил значения. Я сделал так, чтобы максимизировать прибыль (чем больше прибыль за клик, тем больше надо взять среднее количество кликов за день и наоборот).

Время и память

- Время выполнения: 0.000017 секунд (2 секунды ограничение)
- Использовано памяти: 0.000504 Мб

Примеры

Вводный файл	Выходной файл
3 1 3 -5 -2 4 1	23
1 23 39	897

Вывод

Я реализовал сортировку и жадное парное умножение ставок и кликов для максимального дохода, освоил приём жадного алгоритма через предварительную сортировку.

7 задача. Проблема сапожника (0.5 балла)

Постановка задачи

В некоей воинской части есть сапожник. Рабочий день сапожника длится K минут. Заведующий складом оценивает работу сапожника по количеству починенной обуви, независимо от того, насколько сложный ремонт требовался в каждом случае. Дано n сапог, нуждающихся в починке.

Определите, какое максимальное количество из них сапожник сможет починить за один рабочий день.

Листинг кода

```
import time
import tracemalloc

def max_repair(K, repair_times):
    repair_times.sort()

    boots_repaired = 0
    total_time = 0

    for current_time in repair_times:
        if total_time + current_time <= K:
            total_time += current_time
            boots_repaired += 1
        else:
            break

    return boots_repaired

with open("input.txt", "r") as file:
    K, n = map(int, file.readline().strip().split())
    repair_times = list(map(int, file.readline().strip().split()))

start_time = time.perf_counter()
tracemalloc.start()

result = max_repair(K, repair_times)

end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

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

Объяснение кода

Чтобы починить как можно больше сапог, необходимо выбирать сапоги с наименьшим временем починки. Поэтому я отсортировал массив со временем по возрастанию. В цикле я проходился по каждому значению и прибавлял к общему времени починки, сравнивая со временем рабочего дня сапожника. Так получится максимизировать количество починок.

Время и память

- Время выполнения: 0.000011 секунд (2 секунды ограничение)
- Использовано памяти: 0.000046 Мб (256 мб ограничение)

Примеры

Вводный файл	Выходной файл
1 3 2 2 10 20	1 0 2
1 10 3 2 6 2 8	1 2 2

Вывод

Я реализовал сортировку времен ремонта и жадный выбор самых быстрых ремонтов до ограничения K.

9 задача. Распечатка (1 балл)

Постановка задачи

Один лист фирма печатает за A1 рублей, 10 листов - за A2 рублей, 100 листов - за A3 рублей, 1000 листов - за A4 рублей, 10000 листов - за A5 рублей, 100000 листов - за A6 рублей, 1000000 листов - за A7 рублей. При этом не гарантируется, что один лист в более крупном заказе обойдется дешевле, чем в более мелком. И даже может оказаться, что для любой партии будет выгодно воспользоваться тарифом для одного листа. Печать конкретного заказа производится или путем комбинации нескольких тарифов, или путем заказа более крупной партии.

Требуется по заданному объему заказа в листах N определить минимальную сумму денег в рублях, которой будет достаточно для выполнения заказа.

Листинг кода

```
import time, tracemalloc

def min_cost(N, costs):
    size = [1, 10, 100, 1000, 10000, 100000, 1000000]
    min_cost = [float('inf')] * (N + 1)
    min_cost[0] = 0
    for i in range(1, N + 1):
        for j in range(len(size)):
            if i >= size[j]:
                min_cost[i] = min(min_cost[i], min_cost[i - size[j]] +
costs[j])
            else:
                min_cost[i] = min(min_cost[i], costs[j])
    return min_cost[N]

with open("input.txt", "r") as file:
    N = int(file.readline().strip())
    costs = [int(file.readline().strip()) for _ in range(7)]

start_time = time.perf_counter()
tracemalloc.start()
```

```
result = min_cost(N, costs)

end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

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

Объяснение

Данную задачу я решил динамическим программирование. Я создал массив со стоимостями листов, первое значение равно 0, так как это стоимость 0 листов. Затем в цикле я прохожусь по количеству листов и сравниваю с размером партии листов. Если размер больше партии, то я проверяю не выгоднее ли купить эту партию листов. Если же размер листов не превышает размера партии, то сравниваю между двумя значениями стоимости и выбираю наименьшее

Время и память

- Время выполнения: 0.006016 секунд (2 секунды)
- Использовано памяти: 0.028954 Мб (256 мб)

Примеры

Вводный файл	Выходной файл
980 1 9 90 900 1000 10000 10000	882
980 1 10 100 1000 900 10000 10000	900

Вывод

Я реализовал динамическое программирование для поиска минимальной стоимости печати N листов через перебор вариантов партий.

15 задача. Удаление скобок (2 балла)

Постановка задачи

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 time, tracemalloc

def balance_brackets(s):
    stack = []
    index_to_remove = set()
    pairs = {')': '(', ']': '[', '}': '{'}

    for i, char in enumerate(s):
        if char in "([{":
            stack.append((char, i))
        elif char in ")]}":
            if stack and stack[-1][0] == pairs[char]:
                stack.pop()
            else:
                index_to_remove.add(i)

    index_to_remove.update(i for _, i in stack)
    result = ''.join(s[i] for i in range(len(s)) if i not in
index_to_remove)
    return result
with open("input.txt", "r") as file:
    s = file.readline().strip()
start_time = time.perf_counter()
tracemalloc.start()
result = balance_brackets(s)
end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")
with open("output.txt", "w") as file:
    file.write(result + "\n")
```

Объяснение

Я создал список, в котором буду хранить открывающие скобки и его индекс. Следующим я создал множество для хранения индексов, которые нужно удалить, чтобы была правильная скобочная последовательность. Я выбрал множество, потому что в среднем операция в множестве выполняется за O(1) и порядок мне не важен

В цикле я проверяю, если скобка открывающаяся, то добавляю ее в список вместе с его индексом. Если скобка закрывающая, то я проверяю подходит ли последняя добавленная скобка к ней, если подходит, то я удаляю из списка, если нет, то сохраняю индекс. После цикла есть ли скобочные последовательности, которые остались не закрытыми, и если есть, то добавляю их индексы. После чего я прохожусь по строке и удаляю те скобки, индексы которых добавил

Время и память

- Время выполнения: 0.000030 секунд (2 секунды ограничение)
- Использовано памяти: 0.000763 Мб (256 мб ограничение)

Примеры

Вводный файл	Выходной файл
1 ([)] 2	1 [] 2

Вывод

Я реализовал балансировку скобочной последовательности через стек и удаление некорректных символов по индексам.

21 задача. Игра в дурака (3 балла)

Постановка задачи

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 time, tracemalloc

def can_defend(trump_suit, player_cards, attack_cards):
    ranks = {'6': 0, '7': 1, '8': 2, '9': 3, 'T': 4, 'J': 5, 'Q': 6, 'K': 7,
'A': 8}

    player_cards_by_suit = {suit: [] for suit in 'SCDH'}
    for card in player_cards:
        player_cards_by_suit[card[1]].append(card)

    for suit in player_cards_by_suit:
        player_cards_by_suit[suit].sort(key=lambda x: ranks[x[0]])

    for attack_card in attack_cards:
        attack_rank, attack_suit = attack_card[0], attack_card[1]

        possible_defense = [c for c in player_cards_by_suit[attack_suit] if
ranks[c[0]] > ranks[attack_rank]]

        if attack_suit == trump_suit:
            possible_defense = [c for c in possible_defense if c[1] ==
trump_suit]

        if not possible_defense and attack_suit != trump_suit:
            possible_defense = player_cards_by_suit[trump_suit]

        if not possible_defense:
```

```
            return "NO"

        used_card = possible_defense[0]
        player_cards_by_suit[used_card[1]].remove(used_card)

    return "YES"

with open("input.txt", "r") as file:
    first_line = file.readline().strip().split()
    N, M, trump_suit = int(first_line[0]), int(first_line[1]), first_line[2]
    player_cards = file.readline().strip().split()
    attack_cards = file.readline().strip().split()

start_time = time.perf_counter()
tracemalloc.start()

result = can_defend(trump_suit, player_cards, attack_cards)

end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")



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

Объяснение

Сначала создал словарь, где номинал карты - это ключ, а значение - это ее величина. Этот словарь я создал, чтобы удобно было сравнивать номиналы карты. Затем я создаю еще один словарь с картами игрока, где ключ - это масть, а значение - это список карт этой масти, доступные игроку. Затем я добавляю в этот словарь карты игрока и сортирую их, чтобы можно было отбиваться минимально возможной картой. Далее в цикле я прохожусь по каждой атакующей карте и ищу карту у игрока, которой можно отбиться. Если такой карты нет, то значит отбиться не удастся и возвращается NO. Если удалось отбиться, то это карта удаляется у игрока. Если в цикле все карты удалось отбить, то возвращается YES.

Время и память

- Время выполнения: 0.000027 секунд (1 секунда ограничение)
- Использовано памяти: 0.000732 Мб (16 Мб ограничение)

Примеры

Вводный файл	Выходной файл
1 4 1 D 2 9S KC AH 7D 3 8D	1 NO 2
1 6 2 C 2 KD KC AD 7C AH 9C 3 6D 6C	1 YES 2

Проверка на acmp.ru

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
23027918	06.03.2025 14:58:24	Соболев Артём Анатольевич	0698	PyPy	Accepted		0,218	1040 Кб

Вывод

Я сделал группировку и сортировку карт по мастям и рангу и сделал жадное покрытие каждой атакующей карты минимально возможной ответной картой.

ДОПОЛНИТЕЛЬНЫЕ ЗАДАЧИ

1 задача. Максимальная стоимость добычи (0.5 балла)

Постановка задачи

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 time, tracemalloc

def fractional_knapsack(W, items):
    filtered_items = [(v, w, v / w) for v, w in items if w > 0]
    filtered_items.sort(key=lambda x: x[2], reverse=True)

    total_value = 0.0
    for value, weight, ratio in filtered_items:
        if W == 0:
            break
        take = min(weight, W)
        total_value += take * ratio
        W -= take

    return total_value

with open("input.txt", "r") as file:
    n, W = map(int, file.readline().strip().split())
    items = [tuple(map(int, file.readline().strip().split())) for _ in
range(n)]

start_time = time.perf_counter()
tracemalloc.start()

result = fractional_knapsack(W, items)

current_memory, peak_memory = tracemalloc.get_traced_memory()
end_time = time.perf_counter()


print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

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

Объяснение

Я использовал жадный алгоритм. Сначала я отфильтровал все предметы по удельной ценности и потом брал самые выгодные предметы. Если весь предмет не вмещался, то брал столько, сколько есть для этого места в рюкзаке.

Время и память

- Время выполнения: 0.000542 секунд (2 секунды ограничение)
- Использовано памяти: 0.000183 Мб

Примеры

Входной файл	Выходной файл
3 50 60 20 100 50 120 30	180.0000
1 10 500 30	166.6667

Вывод

Я реализовал жадный алгоритм дробного рюкзака через сортировку предметов по удельной ценности и взятие максимально возможной части каждого.

2 задача. Заправки (0.5 балла)

Постановка задачи

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 stop1, stop2, ..., stop из вашего родного города.

Какое минимальное количество заправок необходимо?

Листинг кода

```
import time, tracemalloc

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

    while current_pos < len(stops) - 1:
        last_pos = current_pos
        while (current_pos + 1 < len(stops)) and (stops[current_pos + 1] -
stops[last_pos] <= m):
            current_pos += 1

        if current_pos == last_pos:
            return -1

        if current_pos < len(stops) - 1:
            num_refills += 1

    return num_refills

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

start_time = time.perf_counter()
tracemalloc.start()

result = min_refills(d, m, stops)

current_memory, peak_memory = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

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

Объяснение

Я искал самую дальнюю заправку, до которой можно доехать без подзаправки. Если на каком-то этапе до заправки невозможно было доехать, то возвращается -1, иначе прибавляется счетчик к количеству заправок

Время и память

- Время выполнения: 0.000005 секунд (2 секунды ограничение)
- Использовано памяти: 0.000061 Мб

Примеры

Входной файл	Выходной файл
950 400 4 200 375 550 750	2
10 3 4 1 2 5 9	-1
200 250 2 100 150	0

Вывод

Я реализовал жадный алгоритм заправок через поиск самой дальней станции на каждом шаге.

8 задача. Расписание лекций (1 балл)

Постановка задачи

Есть список заявок от преподавателей на лекции для одной из аудиторий. Каждая заявка представлена в виде временного интервала [s, f) - 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. Одновременно в лекционной аудитории может читаться лишь одна лекция.

Листинг кода

```
import time, tracemalloc
def max_lectures(intervals):
    intervals.sort(key=lambda x: x[1])
    count = 0
    last_end_time = 0
    for start, end in intervals:
        if start >= last_end_time:
            count += 1
            last_end_time = end
    return count
with open("input.txt", "r") as file:
    N = int(file.readline().strip())
    intervals = [tuple(map(int, file.readline().strip().split())) for _ in
range(N)]
start_time = time.perf_counter()
tracemalloc.start()
result = max_lectures(intervals)
end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

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

Объяснение

Так как мне даны интервалы, то я отсортирую интервалы по окончании времени лекции. Это позволит выбирать лекции, которые заканчиваются раньше. Затем в цикле я сравниваю время начала новой лекции со временем окончания предыдущей. Так получится выбрать наибольшее количество лекций.

Время и память

- Время выполнения: 0.000014 секунд (2 секунды ограничение)
- Использовано памяти: 0.000153 Мб (256 мб ограничение)

Примеры

Вводный файл	Выходной файл
3 1 5 2 3 3 4	2
1 5 10	1

Вывод

Я реализовал выбор максимального числа лекций через сортировку по времени окончания и жадный отбор.

17 задача. Ход конём (3 балла)

Постановка задачи

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 time, tracemalloc
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') as f:
    N = int(f.read())
start_time = time.perf_counter()
tracemalloc.start()
dp = [ [0]*10 for _ in range(N+1) ]
for i in range(10):
    if i != 0 and i != 8:
        dp[1][i] = 1
for l in range(2, N+1):
    for d in range(10):
        for prev in moves[d]:
            dp[l][d] = (dp[l][d] + dp[l-1][prev]) % 10**9

result = sum(dp[N]) % 10**9
```

```
end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

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

Объяснение

Для решения задачи я создал матрицу, где первым значением была текущая длина уже набранного номера, а вторым цифра, на которой сейчас нахожусь. Начать набор номера можно со всех цифр, кроме 0 и 8, поэтому у них значения 0 будет. В moves я записал всевозможные ходы коня. После этого я перебираю все цифры, с которых можно попасть ходом в коня в текущую цифру. Так последовательно строится таблица до длины N. После чего все значения суммируются.

Время и память

- Время выполнения: 0.000021 секунд
- Использовано памяти: 0.000259 Мб (2.5 Мб ограничение)

Входной файл	Выходной файл
1	1	1	8
		2	
1	2	1	16
		2	

Вывод

Я реализовал подсчёт номеров ходом коня через динамическое программирование по длине и последней цифре.

18 задача. Кафе (2.5 балла)

Постановка задачи

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 time, tracemalloc

start_time = time.perf_counter()
tracemalloc.start()

with open('input.txt', 'r') as f:
    lines = f.readlines()

n = int(lines[0])
s = [int(x) for x in lines[1:]]

INF = 10**9
dp = [[INF] * (n + 5) for _ in range(n + 1)]
used = [[False] * (n + 5) for _ in range(n + 1)]
prev = [[-1] * (n + 5) for _ in range(n + 1)]

dp[0][0] = 0

for i in range(n):
    for c in range(n + 1):
```

```
        if dp[i][c] == INF:
            continue

        new_c = c + 1 if s[i] > 100 else c
        cost = dp[i][c] + s[i]
        if cost < dp[i + 1][new_c]:
            dp[i + 1][new_c] = cost
            used[i + 1][new_c] = False
            prev[i + 1][new_c] = c

        if c >= 1:
            if dp[i][c] < dp[i + 1][c - 1]:
                dp[i + 1][c - 1] = dp[i][c]
                used[i + 1][c - 1] = True
                prev[i + 1][c - 1] = c
min_cost = INF
final_c = -1
for c in range(n + 1):
    if dp[n][c] < min_cost or (dp[n][c] == min_cost and c > final_c):
        min_cost = dp[n][c]
        final_c = c
coupons_used = []
i, c = n, final_c
while i > 0:
    pc = prev[i][c]
    if used[i][c]:
        coupons_used.append(i)
    i -= 1
    c = pc
coupons_used.sort()
with open('output.txt', 'w') as f:
    f.write(f"{min_cost}\n")
    f.write(f"{final_c} {len(coupons_used)}\n")
    for day in coupons_used:
        f.write(f"{day}\n")

end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")
```

Объяснение

Для решения задачи я использовал динамическое программирование. Я создал массив, где первым значением был номер дня, а вторым количество купонов после этого дня. На каждом шаге выбирается вариант: потратить купон или заплатить и получить купон, если обед стоит больше 100 рублей. В конце выбирается минимальные затраты.

Время и память

- Время выполнения: 0.000603 секунд (2 секунды ограничение)
- Использовано памяти: 0.000772 Мб (64 Мб ограничение)

Примеры

Входной файл	Выходной файл
5 110 40 120 110 60	260 0 2 3 5
3 110 110 110	220 1 1 2

Вывод

Я реализовал динамическое программирование с учётом купонов, хранение состояния и восстановление пути.

20 задача. Почти палиндром (3 балла)

Постановка задачи

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

Листинг кода

```
import time
import tracemalloc

def count_almost_palindromes(N, K, S):
    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
            else:
                if S[i] == S[j]:
                    dp[i][j] = dp[i + 1][j - 1]
                else:
                    dp[i][j] = dp[i + 1][j - 1] + 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().strip()
start_time = time.perf_counter()
tracemalloc.start()
result = count_almost_palindromes(N, K, S)
end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

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

Объяснение

Я создал массив dp, в котором записано минимальное количество изменений букв в подстроке для создания палиндрома. Затем я прошелся по всем подстрокам строки. Если в подстроке 1 буква, то она уже является палиндромом. Если крайние символы не совпадают, то нужно один из них заменить и определить сколько замен нужно для внутренней подстроки. Если крайние символы равны, то количество замен равно количеству замен для внутренней подстроки. Если количество замен для подстроки меньше или равно k, то увеличиваем счетчик.

Время и память

- Время выполнения: 0.000019 секунд
- Использовано памяти: 0.000206 Мб

Примеры

Входной файл	Выходной файл
1 5 1 2 abcde	1 12 2
1 3 3 2 aaa	1 6 2

Вывод

Я реализовал подсчёт почти палиндромов через динамическое программирование по подстрокам с учётом количества несовпадений.

ВЫВОД

В лабораторной работе я реализовал решения задач с применением жадных алгоритмов и динамического программирования