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 14

Выполнила:
Рудникова В.О.
К3143

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №1. Максимальная стоимость добычи [0.5 балла]

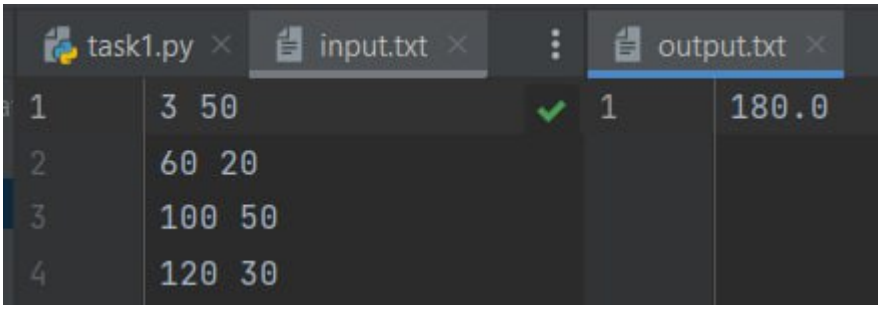
Вор находит гораздо больше добычи, чем может поместиться в его сумке. Помогите ему найти самую ценную комбинацию предметов, предполагая, что любая часть предмета добычи может быть помещена в его сумку. Цель - реализовать алгоритм для задачи о дробном рюкзаке.
Формат ввода / входного файла (input.txt).
В первой строке входных данных задано целое число n - количество предметов, и W - вместимость сумки. Следующие n строк определяют значения веса и стоимости предметов. В i-ой строке содержатся целые числа pi и wi – стоимость и вес i-го предмета, соответственно.
Ограничения на входные данные.
$1 \le n \le 10^3$, $0 \le W \le 2 \cdot 10^6$, $0 \le pi \le 2 \cdot 10^6$, $0 \le wi \le 2 \cdot 10^6$ для всех $1 \le i \le n$. Все числа - целые.
Формат вывода / выходного файла (output.txt). Выведите максимальное значение стоимости долей предметов, которые помещаются в сумку. Абсолютная погрешность между ответом вашей программы и оптимальным значением должно быть не более 10^{-3}. Для этого выведите свой ответ как минимум с четырьмя знаками после запятой (иначе ваш ответ, хотя и будет рассчитан правильно, может оказаться неверным из-за проблем с округлением).

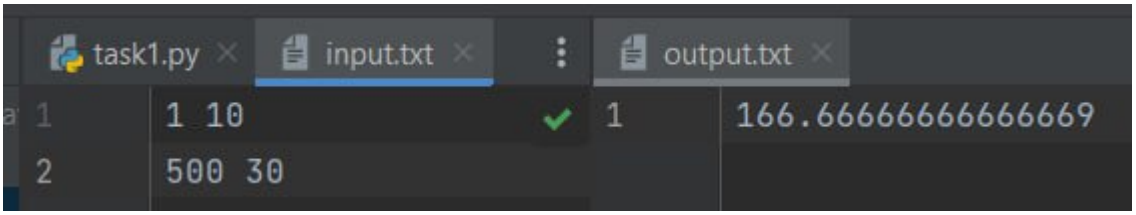
```
from time import process_time
from tracemalloc import start, get_traced_memory


def fractional_knapsack(n: int, w: int, prices: list, weights:
list) -> float:
    ratio = []
    for i in range(n):
        ratio.append(prices[i] / weights[i])
    res = 0
    while w > 0:
        if weights[ratio.index(max(ratio))] > w:
            res += ratio[ratio.index(max(ratio))] * w
            ratio.pop(ratio.index(max(ratio)))
            w = 0
        else:
```

```
            res += ratio[ratio.index(max(ratio))] *
weights[ratio.index(max(ratio))]
            w -= weights[ratio.index(max(ratio))]
            ratio.pop(ratio.index(max(ratio)))

    return res


if __name__ == '__main__':
    start()
    ws, ps = [], []
    with open('input.txt', 'r') as f:
        n, w = map(int, f.readline().split())
        for i in range(n):
            price, weight = map(int, f.readline().split())
            ps.append(price)
            ws.append(weight)
    res = fractional_knapsack(n, w, ps, ws)
    with open('output.txt', 'w+') as g:
        g.write(str(res))

        print('Time:', str(process_time()), 'sec')
        print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения:
Для каждого элемента находится отношение его стоимости к весу и выбираются предметы с максимальным отношением. Если какой-то из них не помещается, берётся его часть.

Результат работы кода на примерах из текста задачи:





Вывод по задаче:
В задаче я реализовала алгоритм решения задачи о дробном рюкзаке.

Задача №5. Максимальное количество призов (0.5 балла)

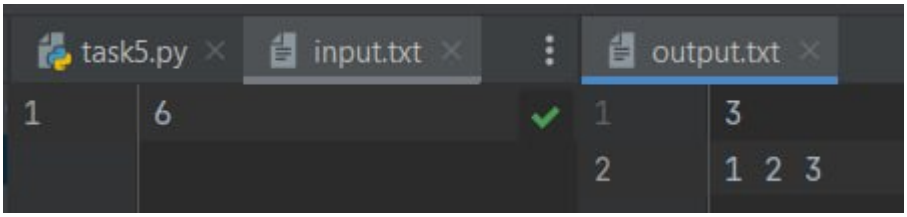
Вы организуете веселый конкурс для детей. В качестве призового фонда у вас есть n конфет. Вы хотели бы использовать эти конфеты для раздачи k лучшим местам в конкурсе с естественным ограничением, заключающимся в том, что чем выше место, тем больше конфет. Чтобы осчастливить как можно больше детей, вам нужно найти наибольшее значение k, для которого это возможно.

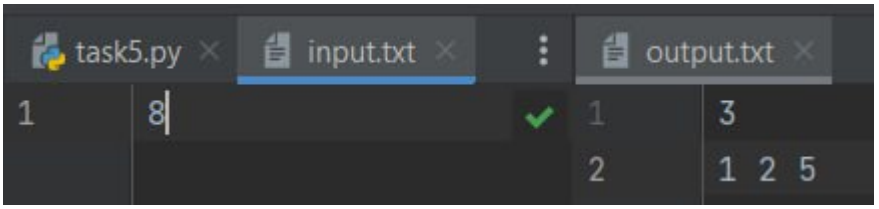
```
from time import process_time
from tracemalloc import start, get_traced_memory


def max_prize(n: int) -> list:
if n == 1:
return [1]
if n == 2:
return [2]
prizes = [0]
while n > max(prizes):
maxx = max(prizes)
if n - (maxx + 1) not in prizes:
prizes.append(maxx + 1)
n -= (maxx + 1)
else:
prizes.append(n)
prizes.remove(0)
return sorted(prizes)


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())
res = max_prize(n)
with open('output.txt', 'w+') as g:
g.write(str(len(res)) + '\n' + ' '.join([str(i) for
i in res]))
```

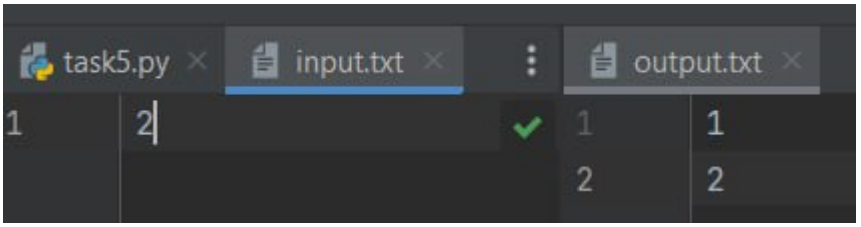
```
print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: последнему из призовых мест мы должны вручить одну конфету, а дальше - вручать на одну больше, проверяя, что первому месту достанется больше всех. Максимальное число, для которого это выполнится, и будет ответом.

Результат работы кода на примерах из текста задачи:







Проверка задачи на (openedu, acmp и тд при наличии в задаче).

Вывод по задаче: в задаче я использовала жадный алгоритм.

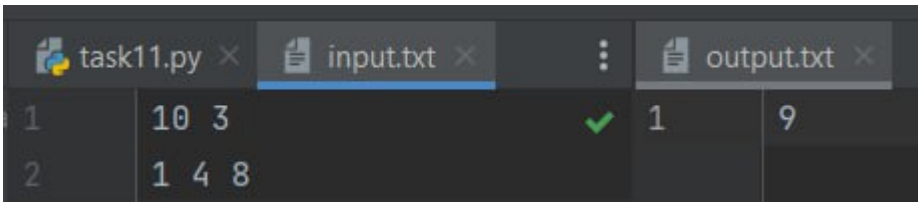
Задача №11. Максимальное количество золота (1 балл)

Вам дается набор золотых слитков, и ваша цель - набрать как можно больше золота в свою сумку. Существует только одна копия каждого слитка, и для каждого слитка вы можете либо взять его, либо нет (т.е. вы не можете взять часть слитка).

```
from time import process_time
from tracemalloc import start, get_traced_memory


def knapsack(n: int, W: int, weights: list[int]) ->
int:
dp = [[0 for _ in range(W + 1)] for _ in range(n +
1)]
for i in range(1, n + 1):
for j in range(1, W + 1):
if weights[i - 1] > j:
dp[i][j] = dp[i - 1][j]
else:
dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weights[i
- 1]] + weights[i - 1])
return dp[n][W]


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
W, n = map(int, f.readline().split())
ws = [int(i) for i in f.readline().split()]
res = knapsack(n, W, ws)
with open('output.txt', 'w+') as g:
g.write(str(res))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: это задача о рюкзаке. Так как предметы делить нельзя, жадностью она решается плохо, я использовала двумерное динамическое программирование, чтобы хранить максимальную возможную цену с учётом веса и количества предметов, а также считать её с помощью ранее полученных значений.

Результат работы кода на примерах из текста задачи:



Вывод по задаче: для решения задачи о рюкзаке я использовала двумерное динамическое программирование.

Задача №16. Продавец (2 балла)

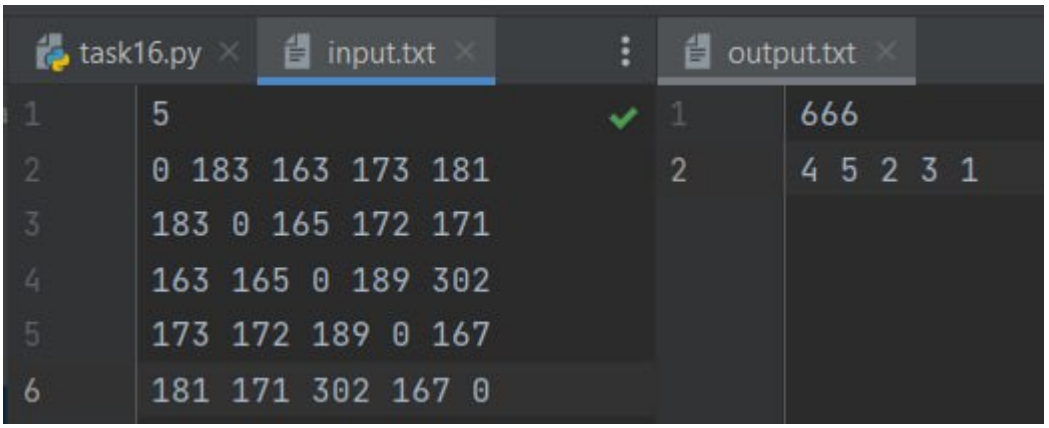
Продавец техники хочет объехать n городов, посетив каждый из них ровно один раз. Помогите ему найти кратчайший путь.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def find_way(arr: list) -> str:
n = int(arr.pop(0))
arr = [list(map(int, line.split())) for line in arr]

visited = [False] * n
lengths = [10**6 + 1] * n
res = [None] * n
curr = 0
lengths[curr] = 0

for i in range(0, n):
min_path_length = 10**6 + 1
for j in range(0, n):
if not visited[j] and lengths[j] < min_path_length:
min_path_length = lengths[j]
curr = j
visited[curr] = True
res[curr] = i + 1

for j in range(0, n):
if not visited[j] and arr[curr][j] < lengths[j]:
lengths[j] = arr[curr][j]

s = str(sum(lengths))
s += "\n"
for i in range(len(res) - 1, -1, -1):
s += str(res[i])
s += " "
return s
```

```
if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
arr = f.readlines()

res = find_way(arr)

with open('output.txt', 'w+') as g:
g.write(res)

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: в задаче мы ищем кратчайший путь с помощью одномерной динамики.

Результат работы кода на примерах из текста задачи:



Вывод по задаче: я использовала одномерную динамику для поиска кратчайшего пути.

Задача №20. Почти палиндром (3 балла)

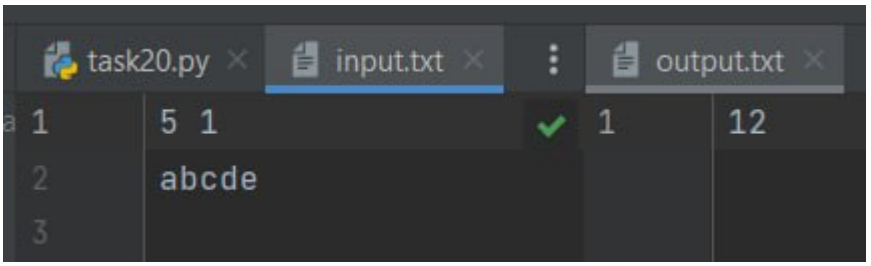
Слово называется палиндромом, если его первая буква совпадает с последней, вторая – с предпоследней и т.д. Например: «abba», «madam», «x».
Для заданного числа K слово называется почти палиндромом, если в нем можно изменить не более K любых букв так, чтобы получился палиндром. Например, при K = 2 слова «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 «c», «a», «t», «ca», «at» 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.

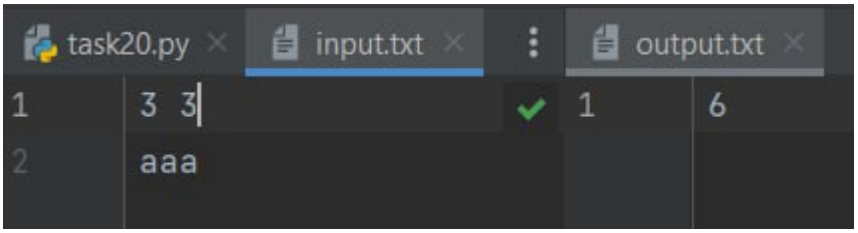
```
from time import process_time
from tracemalloc import start, get_traced_memory


def palindrome(string, n, k):
prev = [0] * n
curr = [0] * n
res = 0
for right in range(n):
res += 1
for left in range(right - 1, -1, -1):
curr[left] = prev[left + 1]
if string[left] != string[right]:
curr[left] += 1
if curr[left] <= k:
res += 1
prev, curr = curr, prev
return res
```

```
if __name__ == '__main__':
start()
with open('input.txt') as f:
n, k = [int(elem) for elem in f.readline().split()]
string = f.readline()

res = palindrome(string, n, k)
with open('output.txt', 'w') as f:
f.write(str(res))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: для поиска почти палиндромов я использовала динамическое программирование, чтобы хранить данные о предыдущих, следующих и нынешних буквах.

Результат работы кода на примерах из текста задачи:





Вывод по задаче: в задаче использовалось одномерное динамическое программирование для нескольких списков.

Дополнительные задачи

Задача №2. Заправки (0.5 балла)

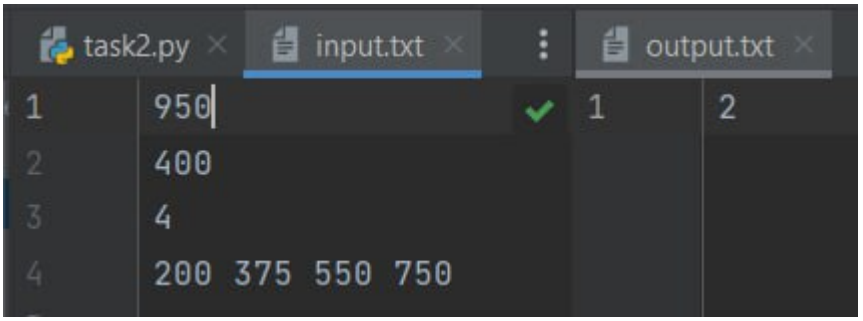
Вы собираетесь поехать в другой город, расположенный в d км от вашего родного города. Ваш автомобиль может проехать не более m км на полном баке, и вы начинаете с полным баком. По пути есть заправочные станции на расстояниях stop1, stop2, ..., stopn из вашего родного города. Какое минимальное количество заправок необходимо?

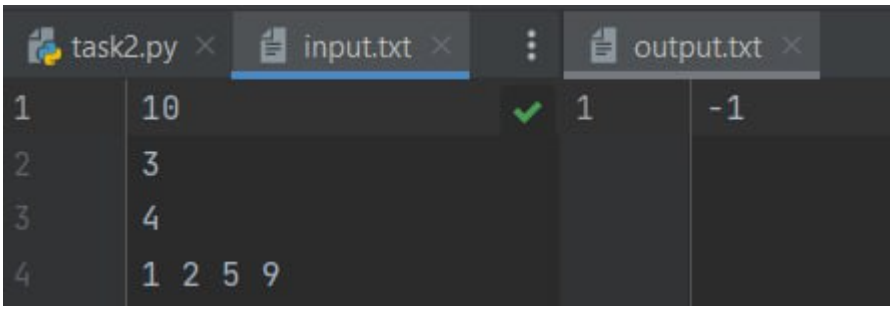
```
from time import process_time
from tracemalloc import start, get_traced_memory


def gas_station(d: int, m: int, n: int, stops: list)
-> int:
used_stops = 0
distance = m
stops.insert(0, 0)
stops.append(d)
for i in range(1, len(stops) - 1):
if stops[i] - stops[i - 1] > m or stops[i + 1] -
stops[i] > m:
return -1
distance -= (stops[i] - stops[i - 1])
if distance < stops[i + 1] - stops[i]:
used_stops += 1
distance = m
return used_stops


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
d = int(f.readline())
```

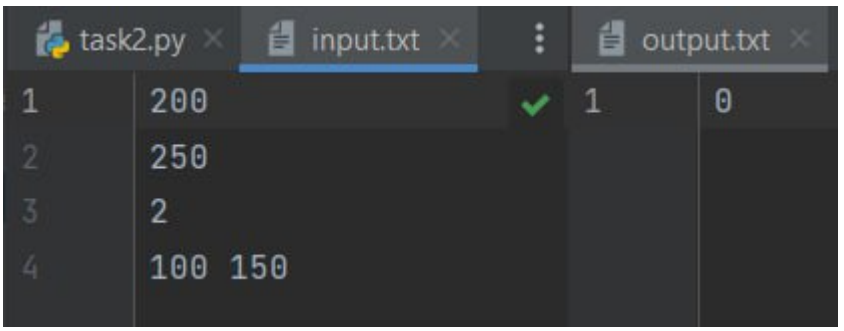
```
m = int(f.readline())
n = int(f.readline())
stops = [int(i) for i in f.readline().split()]
res = gas_station(d, m, n, stops)
with open('output.txt', 'w+') as g:
g.write(str(res))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: в данной задаче был применён жадный алгоритм – едем до самой дальней заправки, до которой можем доехать.

Результат работы кода на примерах из текста задачи:







Вывод по задаче: я использовала жадность для решения задачи о заправках.

Задача №3. Максимальный доход от рекламы (0.5 балла)

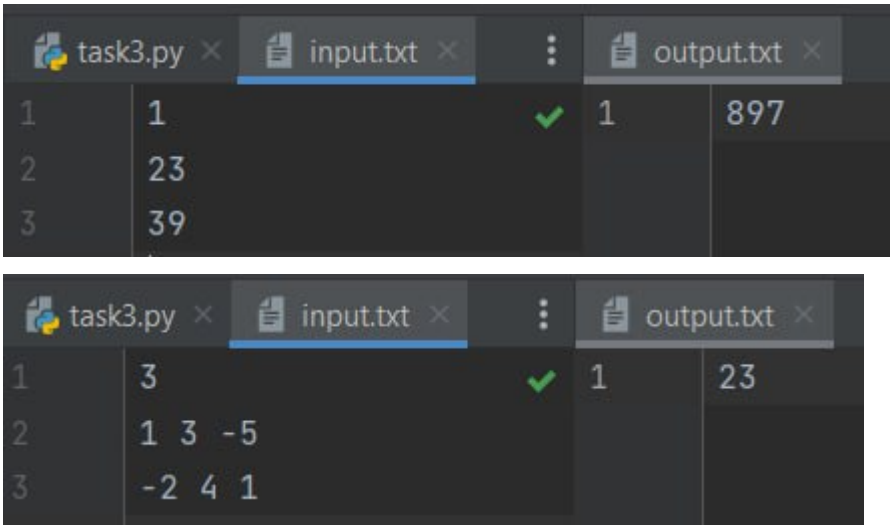
У вас есть n объявлений для размещения на популярной интернет-странице. Для каждого объявления вы знаете, сколько рекламодатель готов платить за один клик по этому объявлению. Вы настроили n слотов на своей странице и оценили ожидаемое количество кликов в день для каждого слота. Теперь ваша цель - распределить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

```
from time import process_time
from tracemalloc import start, get_traced_memory


def max_income(n: int, prices: list, clicks: list)
-> int:
pairs = []
res = 0
for _ in range(n):
pairs.append((max(prices), max(clicks)))
prices[prices.index(max(prices))] = -(10**5) - 1
clicks[clicks.index(max(clicks))] = -(10 ** 5) - 1
for i in pairs:
res += (i[0] * i[1])
return res


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())
a = [int(i) for i in f.readline().split()]
b = [int(i) for i in f.readline().split()]
res = max_income(n, a, b)
with open('output.txt', 'w+') as g:
g.write(str(res))

print('Time:', str(process_time()), 'sec')
```

```
print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')
```

Текстовое объяснение решения: использован жадный алгоритм – самые дорогие слоты нужно размещать так, чтобы кликов по ним было максимальное количество.

Результат работы кода на примерах из текста задачи:



Вывод по задаче: в задаче использовался жадный алгоритм.

Задача №4. Сбор подписей (0.5 балла)

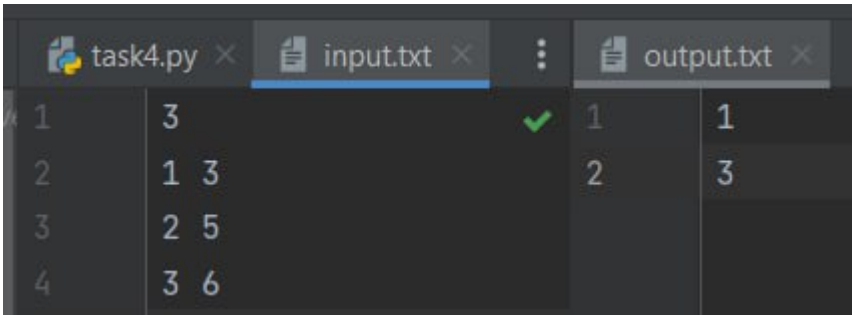
Вы несете ответственность за сбор подписей всех жильцов определенного здания. Для каждого жильца вы знаете период времени, когда он или она находится дома. Вы хотите собрать все подписи, посетив здание как можно меньше раз. Математическая модель этой задачи следующая. Вам дан набор отрезков на прямой, и ваша цель - отметить как можно меньше точек на прямой так, чтобы каждый отрезок содержал хотя бы одну отмеченную точку.

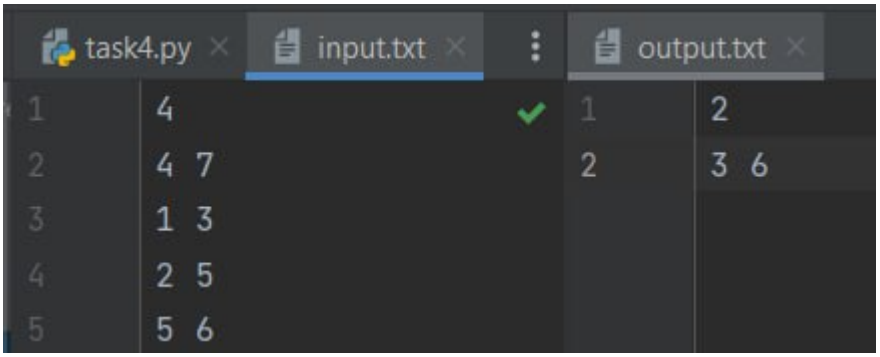
```
from time import process_time
from tracemalloc import start, get_traced_memory


def dots_segments(n: int, segments: list) -> list:
res = 0
dots_used = []
dots_sorted = []
stack = []
covered = [False] * n
for i in segments:
dots_sorted.append((i[0], -1, segments.index(i))) #
левый конец помечен -1
dots_sorted.append((i[1], 1, segments.index(i))) #
правый конец помечен 1
dots_sorted.sort()
for i, dot in enumerate(dots_sorted):
if dot[1] == -1:
stack.append(dot[2])
elif dot[1] == 1:
if not covered[dot[2]]:
res += 1
dots_used.append(dot[0])
for j in stack:
covered[j] = True
stack.pop(-1)
return dots_used
```

```
if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())
segments = []
for i in range(n):
a, b = map(int, f.readline().split())
segments.append((a, b))
res = dots_segments(n, segments)
with open('output.txt', 'w+') as g:
g.write(str(len(res)) + '\n' + ' '.join([str(i) for
i in res]))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: используется жадный алгоритм – точки нужно ставить на такие места, где пересекается больше всего отрезков. Мы ищем такие места и ставим точки так, чтобы их было минимальное количество.

Результат работы кода на примерах из текста задачи:





Вывод по задаче: в задаче использовался жадный алгоритм.

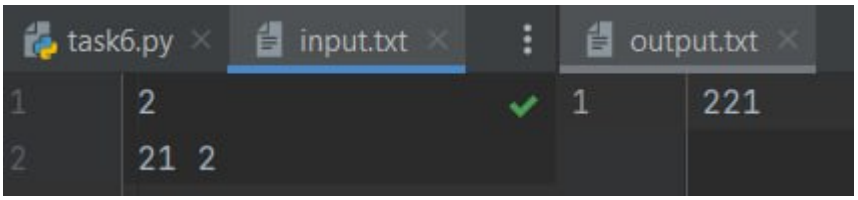
Задача №6. Максимальная зарплата (0.5 балла)

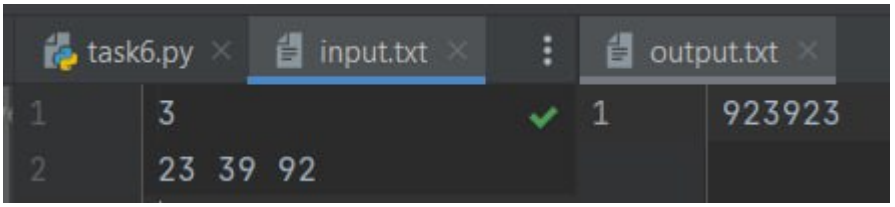
Составить наибольшее число из набора целых чисел.

```
from time import process_time
from tracemalloc import start, get_traced_memory


# функция для сортировки строк по убыванию
def sort_str(a: list[str]) -> list[str]:
for i in range(len(a)):
for j in range(len(a)):
if (a[i] + a[j]) < (a[j] + a[i]) and i < j:
a[i], a[j] = a[j], a[i]
return a


def max_number(nums: list[int]) -> str:
nums = [str(i) for i in nums]
res = ''.join(sort_str(nums))
return res


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())
numbers = [int(i) for i in f.readline().split()]
res = max_number(numbers)
with open('output.txt', 'w+') as g:
g.write(res)

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: это усложнённая версия задачи про составление числа из цифр, но теперь мы учитываем, что числа могут быть не только однозначными, и жадность здесь помогает нам выбрать такое число, которое будет лучше, чем другая комбинация чисел. Для этого пришлось реализовать дополнительную функцию сортировки строк по убыванию.

Результат работы кода на примерах из текста задачи:





Вывод по задаче: я реализовала жадный алгоритм для составления наибольшего числа из набора целых чисел.

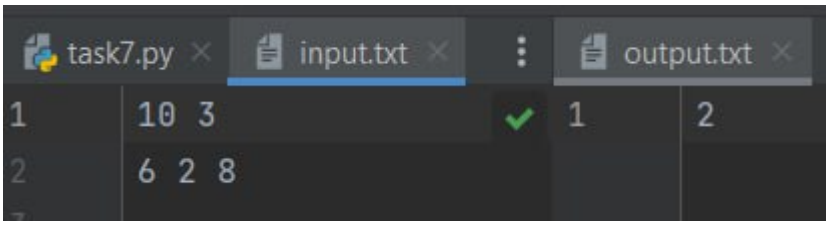
Задача №7. Проблема сапожника (0.5 балла)

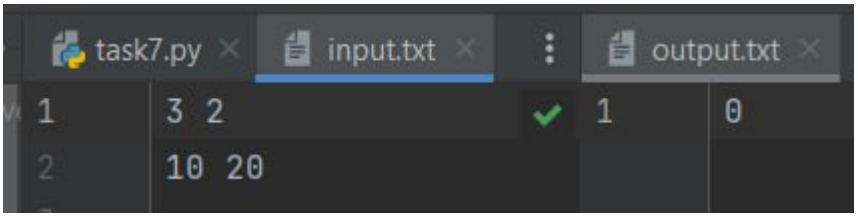
В некоей воинской части есть сапожник. Рабочий день сапожника длится K минут. Заведующий складом оценивает работу сапожника по количеству починенной обуви, независимо от того, насколько сложный ремонт требовался в каждом случае. Дано n сапог, нуждающихся в починке. Определите, какое максимальное количество из них сапожник сможет починить за один рабочий день.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def max_boots(working_time: int, time: list[int]) ->
int:
res = 0
while min(time) < working_time:
minn = min(time)
res += 1
time[time.index(minn)] = 10 ** 3 + 1
working_time -= minn
return res


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
K, n = map(int, f.readline().split())
t = [int(i) for i in f.readline().split()]
res = max_boots(K, t)
with open('output.txt', 'w+') as g:
g.write(str(res))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: использован жадный алгоритм – сначала чинятся наименее сложные сапоги, на которые нужно меньше времени.

Результат работы кода на примерах из текста задачи:





Вывод по задаче: в задаче потребовался жадный алгоритм.

Задача №8. Расписание лекций (1 балл)

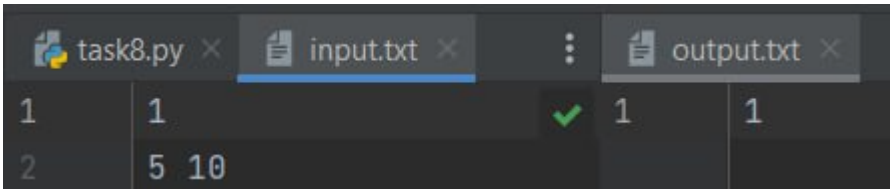
Вы наверно знаете, что в ИТМО лекции читают одни из лучших преподаватели мира. К сожалению, лекционных аудиторий у нас не так уж и много, особенно на Биржевой, поэтому каждый преподаватель составил список лекций, которые он хочет прочитать студентам. Чтобы студенты, в начале февраля, увидели расписание 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 [si , fi) - 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.

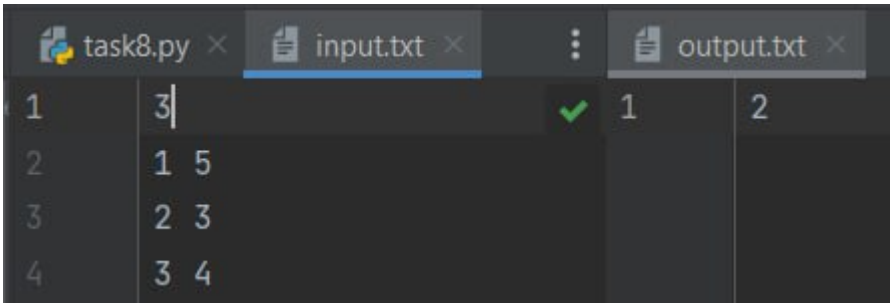
```
from time import process_time
from tracemalloc import start, get_traced_memory


# берём лекцию, у которой конец ближе всего к концу
прошлой (или к началу суток)
# а также проверяем на непересекаемость с другими
(т. е. начало не раньше конца прошлой)
def max_lecs(lectures: list[tuple[int, int]], last:
int, lecs_used: list[int]) -> int:
# возвращаем значение, если список лекций пуст
if len(lectures) == 0:
used = len(lecs_used)
return used
# считаем, насколько конец каждой лекции позже
предыдущей
lecs_diff = [lec[1] - last for lec in lectures]
# если начало какой-то лекции раньше прошлой, она не
проведётся
for i, lec in enumerate(lectures):
if lec[0] < last:
```

```
lecs_diff[i] = 1441
# если ни одну лекцию нельзя провести, возвращаем значение
if lecs_diff.count(1441) == len(lecs_diff):
return len(lecs_used)
# берём лекцию с концом, максимально близким к предыдущей
lecs_used.append(min(lecs_diff) + last)
# обновляем время последней лекции
last += min(lecs_diff)
# удаляем использованную лекцию
lectures.pop(lecs_diff.index(min(lecs_diff)))
return max_lecs(lectures, last, lecs_used)


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())
lectures = []
for i in range(n):
t1, t2 = map(int, f.readline().split())
lectures.append((t1, t2))
res = max_lecs(lectures, 0, [])
with open('output.txt', 'w+') as g:
g.write(str(res))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')
```

Текстовое объяснение решения: нужно выбирать такие лекции, конец у которых наиболее близок к концу прошлых, а также проверять, чтобы они не пересекались. Для этого реализован жадный алгоритм.

Результат работы кода на примерах из текста задачи:





Проверка задачи на (openedu, acmp и тд при наличии в задаче).

Вывод по задаче: здесь снова потребовался жадный алгоритм.

Задача №9. Распечатка (1 балл)

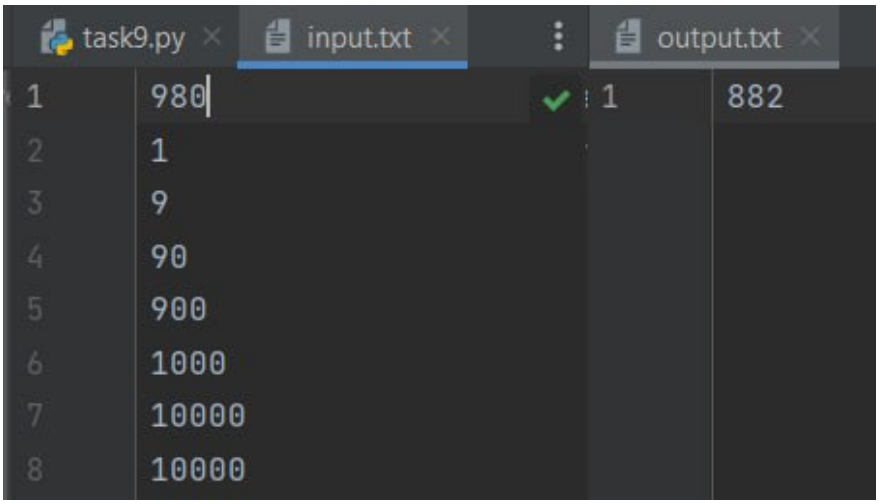
Диссертация дело сложное, особенно когда нужно ее печатать. При этом вам нужно распечатать не только текст самой диссертации, так и другие материалы (задание, рецензии, отзывы, авторефераты для защиты и т.п.). Вы оценили объём печати в N листов. Фирма, готовая размножить печатные материалы, предлагает следующие финансовые условия. Один лист она 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. Например, 980 листов можно распечатать, заказав печать 9 партий по 100 листов плюс 8 партий по 10 листов, сделав 98 заказов по 10 листов, 980 заказов по 1 листу или заказав печать 1000 (или даже 10000 и более) листов, если это окажется выгоднее. Требуется по заданному объему заказа в листах N определить минимальную сумму денег в рублях, которой будет достаточно для выполнения заказа.

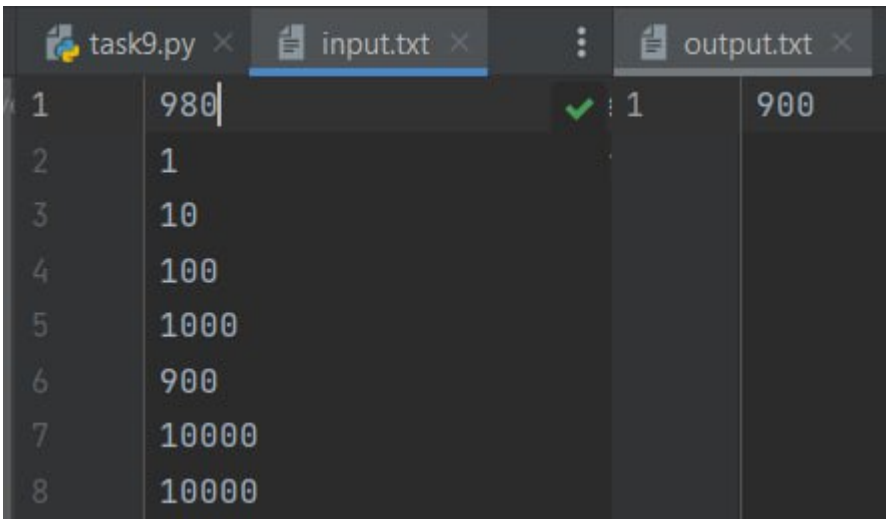
```
from time import process_time
from tracemalloc import start, get_traced_memory


def min_price(n: int, prices: list[int]) -> int:
min_cost = 10 ** 10
result = 0
for i in range(6, -1, -1):
n_new = n
if n != 0:
k = int(n / 10 ** i)
if k > 0:
result += k * prices[i]
n -= k * 10 ** i
if k == 0:
```

```
n_new -= 10**i
if n_new <= 0:
min_cost = min(min_cost, prices[i])
else:
break
return min(result, min_cost)


if __name__ == '__main__':
with open('input.txt', 'r') as f:
N = int(f.readline())
prices = [int(f.readline()) for _ in range(7)]
res = min_price(N, prices)
with open('output.txt', 'w') as g:
g.write(str(res))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: в этой задаче мы используем жадность, чтобы выбрать наиболее удобный способ печати (выбираем самую маленькую стоимость листа и пытаемся купить на неё как можно больше).

Результат работы кода на примерах из текста задачи:





Вывод по задаче: в задаче потребовался жадный алгоритм.

Задача №10. Яблоки (1 балл)

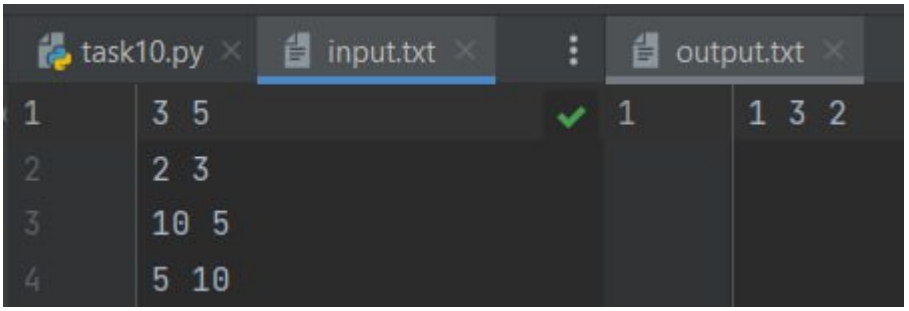
Алисе в стране чудес попались n волшебных яблок. Про каждое яблоко известно, что после того, как его съешь, твой рост сначала уменьшится на ai сантиметров, а потом увеличится на bi сантиметров. Алиса очень голодная и хочет съесть все n яблок, но боится, что в какой-то момент ее рост s станет равным нулю 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.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def apples(a: list[int], b: list[int], s: int, n:
int, order: list[int], start_apples: list[int]) \
-> list[int]:
if n == 0:
return order
diffs = [0] * n
max_diff = 0
for i in range(n):
diffs[i] += (b[i] - a[i])
if s - a[i] > 0:
max_diff = max(max_diff, diffs[i])
else:
diffs[i] = -10 ** 6
if diffs.count(-10 ** 6) == n:
return [-1]
order.append(start_apples.index(max(diffs)) + 1)
s += diffs[diffs.index(max(diffs))]
a.pop(diffs.index(max(diffs)))
b.pop(diffs.index(max(diffs)))
diffs.pop(diffs.index(max(diffs)))
n -= 1
return apples(a, b, s, n, order, start_apples)
```

```
if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n, s = map(int, f.readline().split())
a, b = [0] * n, [0] * n
start_apples = []
for i in range(n):
a[i], b[i] = map(int, f.readline().split())
start_apples.append(b[i] - a[i])
res = apples(a, b, s, n, [], start_apples)
with open('output.txt', 'w+') as g:
g.write(' '.join([str(i) for i in res]))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Текстовое объяснение решения: применяем динамику, смотрим, существует ли яблоко, которое можно съесть в данный момент, чтобы не исчезнуть. Если существует несколько таких, выбираем то, которое максимально увеличит рост или минимально уменьшит.

Результат работы кода на примерах из текста задачи:



Вывод по задаче: здесь потребовалось динамическое программирование.

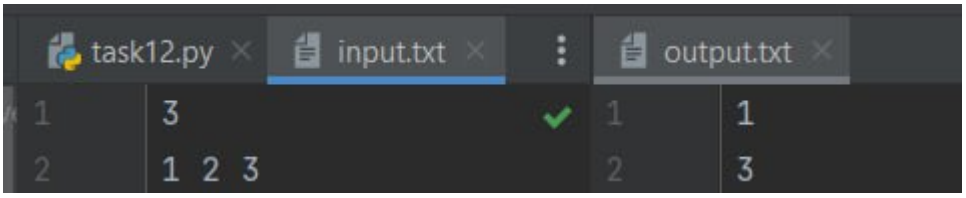
Задача №12. Последовательность (1 балл)

Дана последовательность натуральных чисел a1, a2, ..., an, и известно, что ai ≤ i для любого 1 ≤ i ≤ n. Требуется определить, можно ли разбить элементы последовательности на две части таким образом, что сумма элементов в каждой из частей будет равна половине суммы всех элементов последовательности.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def  two_subsequences(n:  int,  arr:  list[int])  ->
list[int]:
total_sum = sum(arr)
if total_sum % 2 != 0:
return [-1]
half_sum = total_sum // 2
dp  =  [[False]  *  (half_sum  +  1)  for  _  in  range(n  +
1)]
for i in range(n + 1):
dp[i][0] = True
for i in range(1, n + 1):
for j in range(1, half_sum + 1):
if j >= arr[i - 1]:
dp[i][j] = dp[i - 1][j] or dp[i - 1][j - arr[i - 1]]
else:
dp[i][j] = dp[i - 1][j]
if not dp[n][half_sum]:
return [-1]
i, j = n, half_sum
result = []
while i > 0 and j > 0:
if not dp[i-1][j]:
result.append(arr[i-1])
j -= arr[i-1]
i -= 1
```

```
return result


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())
arr = list(map(int, f.readline().split()))
res = two_subsequences(n, arr)
with open('output.txt', 'w+') as g:
if res == [-1]:
g.write("-1\n")
else:
g.write(str(len(res)) + "\n" + " ".join([str(x) for
x in res]))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Результат работы кода на примерах из текста задачи:



Проверка задачи на (openedu, acmp и тд при наличии в задаче).

Вывод по задаче: задача решается с помощью одномерного динамического программирования.

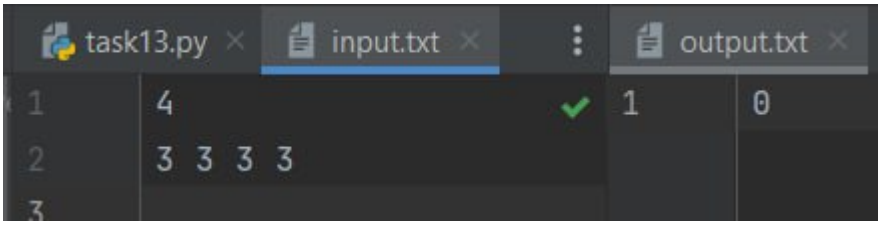
Задача №13. Сувениры (1.5 балла)

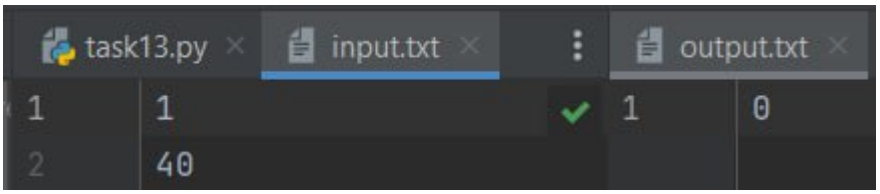
Вы и двое ваших друзей только что вернулись домой после посещения разных стран. Теперь вы хотели бы поровну разделить все сувениры, которые все трое накупили.

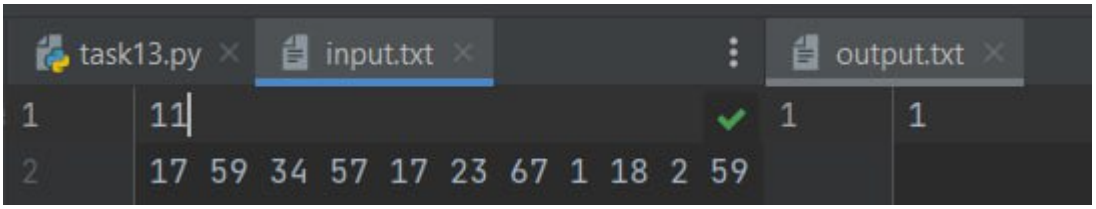
```
from time import process_time
from tracemalloc import start, get_traced_memory


def can_divide(nums: list[int], i: int, subset_sum:
int, subset_count: int, taken: list[bool]) -> bool:
if subset_count == 0:
return True
if subset_sum == 0:
return   can_divide(nums,   0,   sum(nums)   //   3,
subset_count - 1, taken)
for j in range(i, len(nums)):
if not taken[j] and subset_sum - nums[j] >= 0:
taken[j] = True
if  can_divide(nums,  j  +  1,  subset_sum  -  nums[j],
subset_count, taken):
return True
taken[j] = False
return False


def can_divide_equally(nums: list[int]) -> bool:
if len(nums) < 3:
return False
total_sum = sum(nums)
if total_sum % 3 != 0:
return False
subset_sum = total_sum // 3
subset_count = 3
taken = [False for _ in range(len(nums))]
```

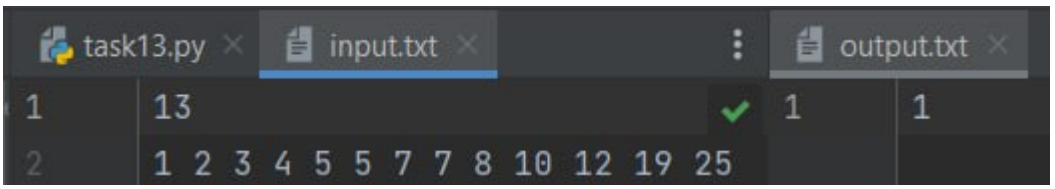
```
return can_divide(nums, 0, subset_sum, subset_count,
taken)


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())
nums = list(map(int, f.readline().split()))

with open('output.txt', 'w') as f:
f.write(str(int(can_divide_equally(nums))))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Результат работы кода на примерах из текста задачи:









Вывод по задаче: задача решается с помощью одномерной динамики.

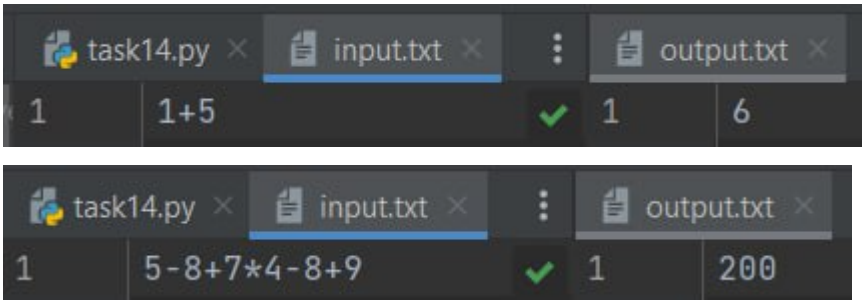
Задача №14. Максимальное значение арифметического выражения (2 балла)

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

```
from time import process_time
from tracemalloc import start, get_traced_memory


def max_result(nums: list[int], ops: list[str]) ->
int:
n = len(nums)
dp_max = [[-10**6] * n for _ in range(n)]
dp_min = [[10**6] * n for _ in range(n)]
for i in range(n):
dp_max[i][i] = dp_min[i][i] = nums[i]
for length in range(2, n + 1):
for i in range(n - length + 1):
j = i + length - 1
for k in range(i, j):
if ops[k] == '+':
dp_max[i][j] = max(dp_max[i][j], dp_max[i][k] +
dp_max[k+1][j])
dp_min[i][j] = min(dp_min[i][j], dp_min[i][k] +
dp_min[k+1][j])
elif ops[k] == '-':
dp_max[i][j] = max(dp_max[i][j], dp_max[i][k] -
dp_min[k+1][j])
dp_min[i][j] = min(dp_min[i][j], dp_min[i][k] -
dp_max[k+1][j])
else:
dp_max[i][j] = max(dp_max[i][j], dp_max[i][k] *
dp_max[k+1][j], dp_min[i][k] * dp_max[k+1][j],
dp_max[i][k] * dp_min[k+1][j], dp_min[i][k] *
dp_min[k+1][j])
```

```
dp_min[i][j] = min(dp_min[i][j], dp_min[i][k] *
dp_min[k+1][j], dp_min[i][k] * dp_max[k+1][j],
dp_max[i][k] * dp_min[k+1][j], dp_max[i][k] *
dp_max[k+1][j])
return dp_max[0][n-1]


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
s = f.readline()

nums = []
ops = []
for i in range(len(s)):
if i % 2 == 0:
nums.append(int(s[i]))
else:
ops.append(s[i])

res = max_result(nums, ops)
with open('output.txt', 'w+') as g:
g.write(str(res) + "\n")

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Результат работы кода на примерах из текста задачи:



Вывод по задаче: я использую двумерную динамику из трёх списков, чтобы найти максимальное значение выражения.

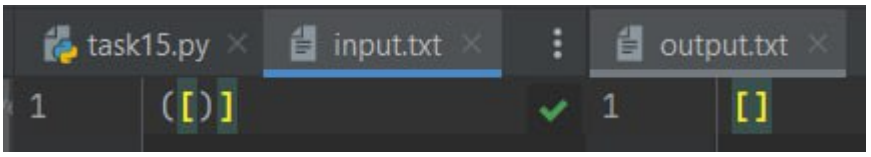
Задача №15. Удаление скобок (2 балла)

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.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def modify_brackets(seq):
n = len(seq)
vals = [[0] * n for i in range(n)]
pos = [[0] * n for i in range(n)]
for right in range(n):
for left in range(right, -1, -1):
if right == left:
vals[left][right] = 1
else:
min_value = 10 ** 9
min_pos = -1
if (seq[left] == '(' and seq[right] == ')') or
(seq[left] == '[' and seq[right] == ']') or (
seq[left] == '{' and seq[right] == '}'):
min_value = vals[left + 1][right - 1]
for k in range(left, right):
if min_value > vals[left][k] + vals[k + 1][right]:
min_value = vals[left][k] + vals[k + 1][right]
min_pos = k
vals[left][right] = min_value
pos[left][right] = min_pos
return vals, pos


def output(l, r, res, vals, pos):
if vals[l][r] == r - l + 1:
```

```
return res
if vals[l][r] == 0:
res += arr[l:r + 1]
return res
if pos[l][r] == -1:
res += arr[l]
res = output(l + 1, r - 1, res, vals, pos)
res += arr[r]
return res
res = output(l, pos[l][r], res, vals, pos)
res = output(pos[l][r] + 1, r, res, vals, pos)
return res


if __name__ == '__main__':
start()
with open('input.txt') as f:
arr = f.readline()
values, position = modify_brackets(arr)
with open('output.txt', 'w+') as g:
g.write(output(0,   len(arr)   -   1,   '',   values,
position))

print('Time:', str(process_time()), 'sec')
print('Memory  usage:',  str(get_traced_memory()[1]  /
1024), 'KB')
```

Результат работы кода на примерах из текста задачи:



Вывод по задаче: здесь я использовала двумерную динамику, чтобы отслеживать удаляемые скобки и контролировать, чтобы скобочная последовательность получалась правильной.

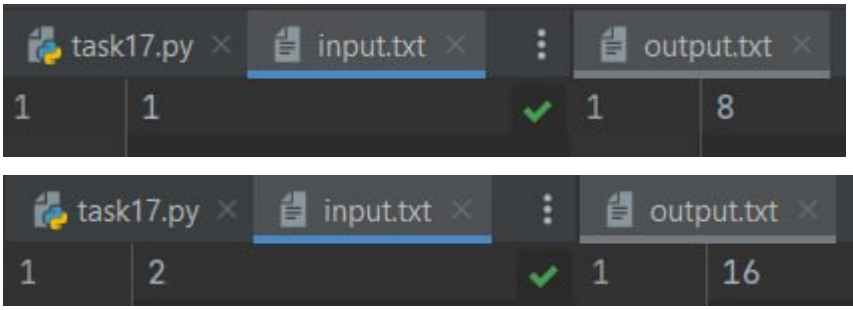
Задача №17. Ход конем (2.5 балла)

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 109.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def hod_konem(n):
prev = [4, 2, 1, 0]
curr = [0, 0, 0, 0]
if n <= 1:
return 8
for i in range(0, n - 1):
curr[0] += prev[1] * 2 + prev[2] * 2
curr[1] += prev[0] + prev[3] * 2
curr[2] += prev[0]
curr[3] += prev[1]
prev = []
for i in range(0, 4):
prev.append(curr[i])
curr = [0, 0, 0, 0]
return sum(prev) % 10**9


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())

res = hod_konem(n)
```

```
with open('output.txt', 'w+') as g:
g.write(str(res))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Результат работы кода на примерах из текста задачи:



Вывод по задаче: для решения задачи используется динамика, чтобы считать количество номеров заданной длины исходя из количества номеров длины на один меньше.

Задача №18. Кафе (2.5 балла)

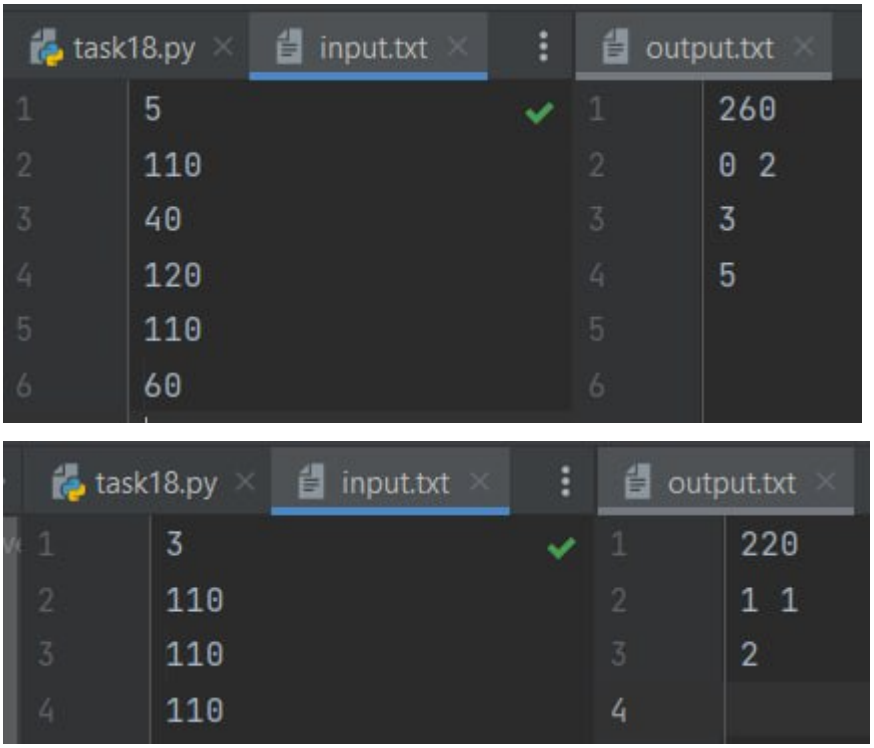
Около университета недавно открылось новое кафе, в котором действует следующая система скидок: при каждой покупке более чем на 100 рублей покупатель получает купон, дающий право на один бесплатный обед (при покупке на сумму 100 рублей и меньше такой купон покупатель не получает). Однажды вам на глаза 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.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def find_ks(n, trans, prevs, free_days):
ans = 10**9 + 1
k1 = 0
for i in range(n + 1):
if trans[n][i] != -1 and trans[n][i] <= ans:
ans = trans[n][i]
k1 = i
j = k1
k2 = 0
for i in range(n, 0, -1):
if j + 1 == prevs[i][j]:
k2 += 1
free_days[k2] = i
j = prevs[i][j]
res = []
res += str(ans) + "\n"
res += str(k1) + " " + str(k2) + "\n"
for i in range(k2, 0, -1):
```

```
res += str(free_days[i]) + "\n"
return res


def lunch_cost(n, arr):
trans = [[-1] * (n + 1) for i in range(n + 1)]
trans[0][0] = 0
prevs = [[-1] * (n + 1) for i in range(n + 1)]
free_days = [0] * (n + 1)
for i in range(n):
for j in range(i + 1):
if trans[i][j] != -1:
if j > 0:
if trans[i + 1][j - 1] > trans[i][j] or trans[i +
1][j - 1] == -1:
trans[i + 1][j - 1] = trans[i][j]
prevs[i + 1][j - 1] = j
k = 0
if arr[i + 1] > 100:
k = 1
if trans[i + 1][j + k] > trans[i][j] + arr[i + 1] or
trans[i + 1][j + k] == -1:
trans[i + 1][j + k] = trans[i][j] + arr[i + 1]
prevs[i + 1][j + k] = j
return find_ks(n, trans, prevs, free_days)


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())
arr = [0]
temp = f.readlines()
for i in range(n):
arr.append(int(temp[i]))

res = lunch_cost(n, arr)
```

```
with open('output.txt', 'w+') as g:
g.write(''.join(map(str, res)))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Результат работы кода на примерах из текста задачи:



Вывод по задаче: я использовала динамику, чтобы вычислить наиболее выгодную конфигурацию абонемента на обеды.

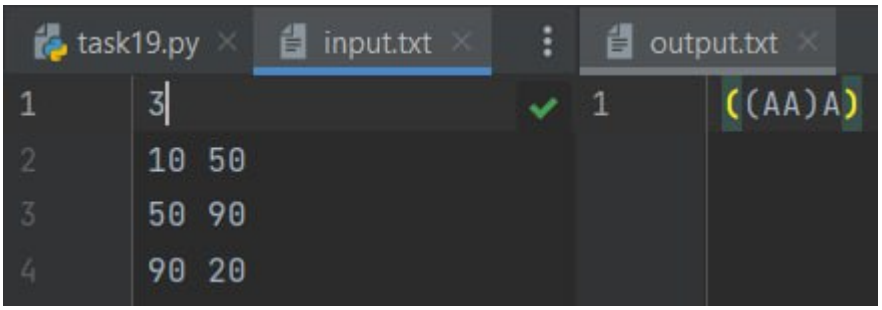
Задача №19. Произведение матриц (3 балла)

В произведении последовательности матриц полностью расставлены скобки, если выполняется один из следующих пунктов:
– Произведение состоит из одной матрицы.
– Оно является заключенным в скобки произведением двух произведений с полностью расставленными скобками.
Полная расстановка скобок называется оптимальной, если количество операций, требуемых для вычисления произведения, минимально. Требуется найти оптимальную расстановку скобок в произведении последовательности матриц.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def output(path, i, j):
if i == j:
return 'A'
return   '('   +   output(path,   i,   path[i][j])   +
output(path, path[i][j] + 1, j) + ')'


def multiply(n, arr):
max = 10 ** 9 + 1
dp = [[0 for i in range(n + 1)] for j in range(n +
1)]
res = [[0 for i in range(n + 1)] for j in range(n +
1)]
for length in range(2, n + 1):
for i in range(1, n - length + 2):
j = i + length - 1
dp[i][j] = max
for k in range(i, j):
cost = dp[i][k] + dp[k + 1][j] + arr[i - 1] * arr[k]
* arr[j]
if cost < dp[i][j]:
```

```
dp[i][j] = cost
res[i][j] = k
return output(res, 1, n)


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n = int(f.readline())
arr = [0 for _ in range(n + 1)]
for index, matrix in enumerate(f.readlines()):
arr[index], arr[n] = tuple(map(int,
matrix.strip().split()))

with open('output.txt', 'w+') as g:
g.write(str(multiply(n, arr)))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Результат работы кода на примерах из текста задачи:



Вывод по задаче: для оптимального расставления скобок при перемножении матриц я использовала динамику, чтобы считать конфигурацию скобок относительно предыдущей (то есть той, где матриц было меньше, и про которую мы поняли, что она оптимальная).

Задача №21. Игра в дурака (3 балла)

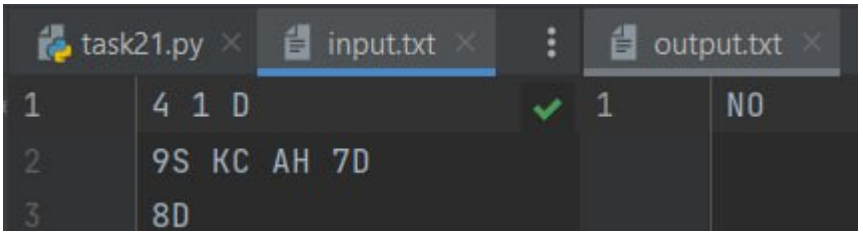
Петя очень любит программировать. Недавно он решил реализовать популярную карточную игру «Дурак». Но у Пети пока маловато опыта, ему срочно нужна Ваша помощь. Как известно, в «Дурака» играют колодой из 36 карт. В Петиной программе каждая карта представляется в виде строки из двух символов, где первый 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.

```
from time import process_time
from tracemalloc import start, get_traced_memory


def play(own, opposite, trump):
opposite_num = len(opposite)
deck = ['6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A',
trump]
cnt, i, j = 0, 0, 0
own.sort(key=lambda     x:     (x[1]     ==     trump,
deck.index(x[0])))
while i < len(own):
while j < len(opposite):
if len(opposite) == 0:
break
elif own[i][1] == opposite[j][1]:
if                deck.index(own[i][0])                >
deck.index(opposite[j][0]):
cnt += 1
own.pop(i)
```

```
i = max(i - 1, 0)
opposite.pop(j)
j -= 1
else:
if own[i][1] == trump:
if opposite[j][1] != trump:
cnt += 1
own.pop(i)
i = max(i - 1, 0)
opposite.pop(j)
j -= 1
else:
if                deck.index(own[i][0])                >
deck.index(opposite[j][0]):
cnt += 1
own.pop(i)
i = max(i - 1, 0)
opposite.pop(j)
j -= 1
j += 1
i = max(i + 1, 0)
j = 0
return cnt, own, opposite_num


if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
n1, n2, trump = tuple(f.readline().split())
own = f.readline().split()
opposite = f.readline().split()

res = play(own, opposite, trump)

with open('output.txt', 'w+') as g:
if res[0] == res[-1]:
g.write('YES')
else:
```

```
g.write('NO')

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Результат работы кода на примерах из текста задачи:



Вывод по задаче: я использовала динамику для того, чтобы запоминать ходы противника в “дураке” и свою колоду.

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 × 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.

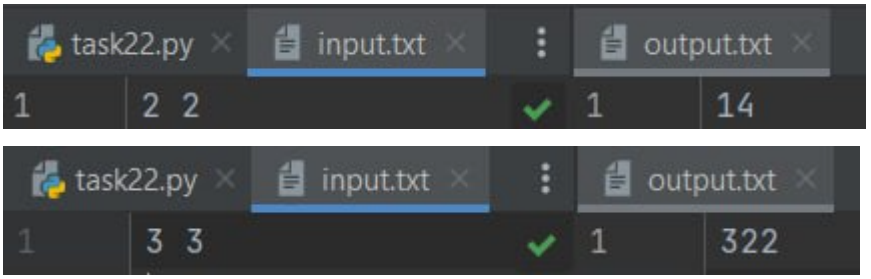
Для составления финансового плана директору необходимо узнать, сколько клиентов он сможет обслужить, прежде чем симпатичные узоры данного размера закончатся. Помогите ему!

```
from time import process_time
from tracemalloc import start, get_traced_memory


def square(x, y):
for i in range(len(x)):
x[i] += y[i]
for i in range(1, len(x)):
if x[i] == 0 and x[i - 1] == 0:
return False
if x[i] == 2 and x[i - 1] == 2:
return False
return True


def field(h):
p = 2
for i in range(1, h):
p *= 2
```

```
f = []
for i in range(0, p):
temp = bin(i)[2:]
temp = temp[::-1]
while len(temp) < h:
temp += "0"
arr = []
for i in range(0, h):
arr.append(int(temp[i]))
f.append(arr)
res = []
for i in f:
res.append([])
for j in f:
if square(i, j):
res[-1].append(1)
else:
res[-1].append(0)
return res


def solve(m, n):
h = min(n, m)
w = max(n, m)
f = field(h)
res = [1] * len(f)
temp = [0] * len(f)
for i in range(0, w - 1):
for j in range(0, len(f)):
for k in range(len(f[j])):
if f[j][k] == 1:
temp[j] += res[k]
for j in range(0, len(f)):
res[j] = temp[j]
temp = [0] * len(f)
return sum(res)
```

```
if __name__ == '__main__':
start()
with open('input.txt', 'r') as f:
m, n = map(int, f.readline().split())

res = solve(m, n)

with open('output.txt', 'w+') as g:
g.write(str(res))

print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] /
1024), 'KB')
```

Результат работы кода на примерах из текста задачи:



Проверка задачи на (openedu, acmp и тд при наличии в задаче).

Вывод по задаче: динамика использована, чтобы считать симпатичность узоров данного размера исходя из более маленьких узоров.

Вывод

В этой работе я научилась решать задачи с помощью жадных алгоритмов и повторила принципы динамического программирования.