САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Жадные алгоритмы. Динамическое программирование
Вариант 5

Выполнил:
Егоров Алексей Алексеевич
К3141

Проверила:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №2. Заправки [0.5 баллов]

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 $stop_1, stop_2, ..., stop_n$ из вашего родного города. Какое минимальное количество заправок необходимо?

- **Формат ввода / входного файла (input.txt).** В первой строке содержится d - целое число. Во втройо строке - целое число m. В третьей находится количество заправок на пути - n. И, наконец, в последней строке - целые числа через пробел - остановки $stop_1, stop_2, ..., stop_n$.
- **Ограничения на входные данные.** $1 \leq d \leq 10^5$, $1 \leq m \leq 400$, $1 \leq n \leq 300$, $1 < stop_1 < stop_2 < ... < stop_n < d$
- **Формат вывода / выходного файла (output.txt).** Предполагая, что расстояние между городами составляет d км, автомобиль может проехать не более m км на полном баке, а заправки есть на расстояниях $stop_1, stop_2, ..., stop_n$ по пути, *выведите минимально необходимое количество заправок*. Предположим, что машина начинает ехать с полным баком. Если до места назначения добраться невозможно, выведите -1.
- Ограничение по времени. 2 сек.
- Примеры:

input.txt	output.txt
950 400 4 200 375 550 750	2

В первом примере расстояние между городами 950 км, на полном баке машина может проехать максимум 400 км. Достаточно сделать две заправки: в точках 375 и 750. Это минимальное количество заправок, так как при одной заправке можно проехать не более 800 км.

input.txt	output.txt	input.txt	output.txt
10 3 4 1 2 5 9	-1	200 250 2 100 150	0

Во втором примере до заправки в точке 9 добраться нельзя, так как предыдущая заправка слишком далеко.

В последнем примере нет необходимости заправлять бак, так как автомобиль стартует с полным баком и может проехать 250 км, а расстояние до пункта назначения составляет 200 км.

Листинг кода

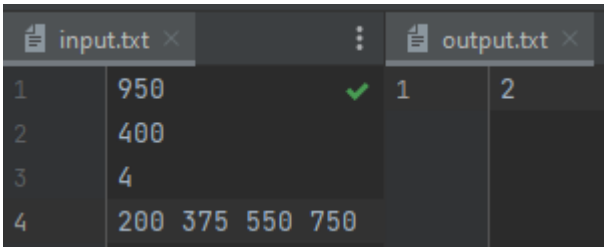
```
import time
import tracemalloc
```

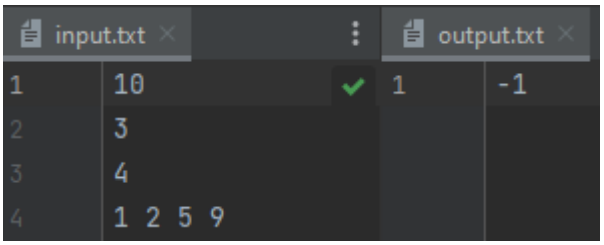
```
def min_count_stops(x, d, m, n):
    x = [0] + x + [d]
    result = 0
    current_stops = 0
    while current_stops <= n:
        last_stops = current_stops
        while current_stops <= n and x[current_stops + 1] - x[last_stops] <= m:
            current_stops += 1
        if last_stops == current_stops:
            return -1
        if current_stops <= n:
            result += 1
    return result


def main():
    with open('input.txt') as f:
        d = int(f.readline().strip())
        m = int(f.readline().strip())
        n = int(f.readline().strip())
        stop = [int(i) for i in f.readline().split()]
    with open("output.txt", "w") as f:
        f.write(str(min_count_stops(stop, d, m, n)))


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

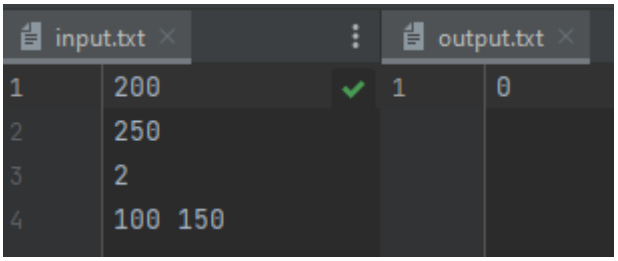
Текстовое объяснение решения

После считывания данных, добавляем к массиву заправок [0] и [d], Они выступают начальными и финальными точками. После этого, идет движение по массиву заправок, пытаясь найти самую дальнюю из возможных, если такую найти не удаётся, выводится -1.

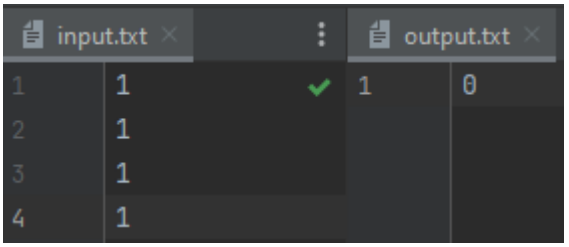
Результат работы кода на примерах из текста задачи:

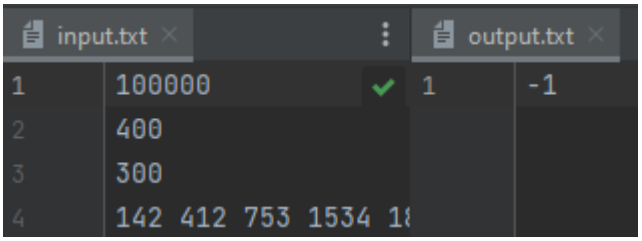






Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница	0.000324600143358111	28,10

диапазона значений входных данных из текста задачи	4	
Пример из задачи	0.0004016000311821699	28,24
Пример из задачи	0.0003737001679837703 7	28,17
Пример из задачи	0.0003551999107003212	28,17
Верхняя граница диапазона значений входных данных из текста задачи	0.0034659953486723785	28,98

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

Задача №4. Сбор подписей [0.5 баллов]

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.

- **Постановка задачи.** Дан набор из n отрезков $[a_0, b_0], [a_1, b_1], ..., [a_{n-1}, b_{n-1}]$ с координатами на прямой, найдите минимальное количество m точек такое, чтобы каждый отрезок содержал хотя бы одну точку. То есть найдите набор целых чисел X минимального размера такой, чтобы для любого отрезка $[a_i, b_i]$ существовала точка $x \in X$ такая, что $a_i \leq x \leq b_i$.
- **Формат ввода / входного файла (input.txt).** Первая строка входных данных содержит количество отрезков n. Каждая из следующих n строк содержит два целых числа a_i и b_i (через пробел), определяющие координаты концов i-го отрезка.
- **Ограничения на входные данные.** $1 \leq n \leq 10^2$, $0 \leq a_i, b_i \leq 10^9$ - целые для всех $1 \leq i \leq n$.
- **Формат вывода / выходного файла (output.txt).** Выведите минимальное количество m точек в первой строке и целочисленные координаты этих m точек (через пробел) во второй строке. Вывести точки можно в любом порядке. Если таких наборов точек несколько, можно вывести любой набор. (Нетрудно видеть, что всегда существует множество точек минимального размера, для которых все координаты точек - целые числа.)
- Ограничение по времени. 2 сек.
- Примеры:

№	input.txt	output.txt
1	3 1 3 2 5 3 6	1 3

№	input.txt	output.txt
2	4 4 7 1 3 2 5 5 6	2 3 6

В первом примере у нас есть три отрезка: $[1, 3], [2, 5], [3, 6]$ (длиной $2, 3, 3$ соответственно). Все они содержат точку с координатой 3: $1 \leq 3 \leq 3$, $2 \leq 3 \leq 5, 3 \leq 3 \leq 6$.

Во втором примере, второй и третий отрезки содержат точку с координатой 3, а первый и четвертый отрезки содержат точку с координатой 6. Все четыре отрезка не могут быть покрыты одной точкой, так как отрезки $[1, 3]$ и $[5, 6]$ не пересекаются.

Листинг кода

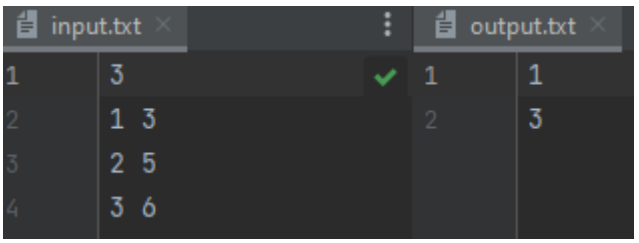
```
import time
import tracemalloc


def main():
```

```
    with open("input.txt", "r") as f:
        n = int(f.readline())
                lr = sorted([tuple(map(int, f.readline().split())) for _ in range(n)],
key=lambda x: x[1])

    count = 0
    sections = []

    for section in lr:
        left, right = section
        if not sections or left > sections[-1]:
            sections.append(right)
            count += 1

    with open("output.txt", "w") as f:
        f.write(f"{count}\n")
        for a in sections:
            f.write(f"{a} ")


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

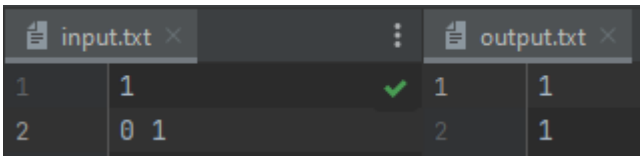
Текстовое объяснение решения

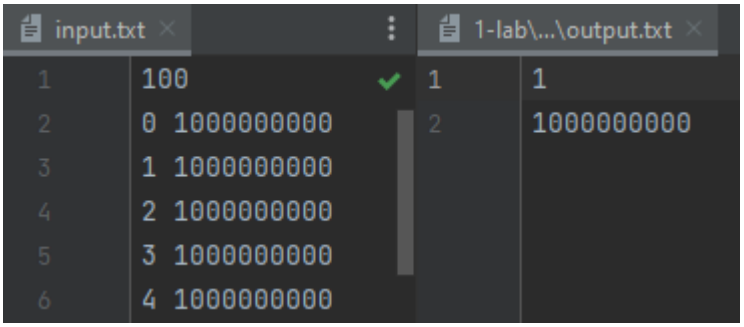
После считывания данных программа сортирует промежутки по правой границе. После этого программа проходит весь массив отрезков, и в случае если следующий отрезок не попадает в нынешнюю точку, то добавляет еще одну точку.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0003417001571506262	18,54
Пример из задачи	0.0003861000295728445	18,99
Пример из задачи	0.0004386999644339084 6	18,54
Верхняя граница диапазона значений входных данных из текста задачи	0.0012558999005705118	23,78

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

Задача №11. Максимальное количество золота [1 баллов]

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).

- **Постановка задачи.** Даны n золотых слитков, найдите максимальный вес золота, который поместится в сумку вместимостью W.
- **Формат ввода / входного файла (input.txt).** Первая строка входных данных содержит вместимость W сумки и количество n золотых слитков. В следующей строке записано n целых чисел $w_0, w_1, ..., w_{n-1}$, определяющие вес золотых слитков.
- **Ограничения на входные данные.** $1 \leq W \leq 10^4$, $1 \leq n \leq 300$, $0 \leq w_0, ..., w_{n-1} \leq 10^5$
- **Формат вывода / выходного файла (output.txt).** Выведите максимальный вес золота, который поместится в сумку вместимости W.
- Ограничение по времени. 5 сек.
- Пример:

input.txt	output.txt
10 3 1 4 8	9

Здесь сумма весов первого и последнего слитка равна 9.

- Обратите внимание, что в этой задаче все предметы имеют одинаковую стоимость на единицу веса по простой причине: все они сделаны из золота.

Листинг кода

```
import time
import tracemalloc


def max_count_gold(capacity, weights):
    dp = [1] + [0] * capacity

    max_index = 0
    for weight in weights:
        index_weight = capacity
        while index_weight - weight >= 0:
            if dp[index_weight - weight] == 1:
                dp[index_weight] = 1
                max_index = max(max_index, index_weight)
            index_weight -= 1
    return max_index
```

```
def main():
    with open('input.txt') as f:
        capacity, n = map(int, f.readline().split())
        weights = [int(weight) for weight in f.readline().split()]

    with open("output.txt", "w") as f:
        f.write(str(max_count_gold(capacity, weights)))


if __name__ == '__main__':
    tracemalloc.start()
    t_start = time.perf_counter()

    main()

    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

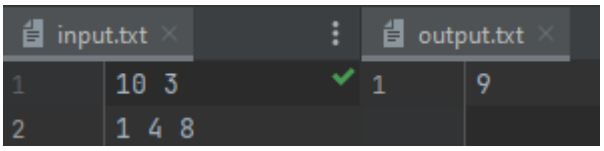
Текстовое объяснение решения

В списке dp размера (W + 1), где W – вместимость рюкзака, индексы элементов соответствуют гипотетически возможным весам, которые могут быть помещены в рюкзак. Список заполняется нулями, за исключением нулевого элемента – он заполняется цифрой 1, что значит то, что вес, равный нулю, мы можем собрать по умолчанию.

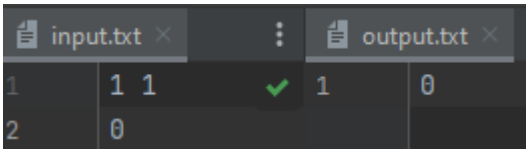
Далее мы выясняем, можно ли с помощью имеющихся слитков набрать ту или иную сумму от 1 до W. Во внешнем цикле перебираются все доступные нам слитки золота из списка weights. Во внутреннем цикле проверяется, можно ли добрать конкретный вес с помощью доступного слитка. Текущий вес можно набрать тогда, когда можно набрать вес, который меньше текущего на вес данного слитка золота. Если его можно набрать, то элементу списка dp присваивается единица.

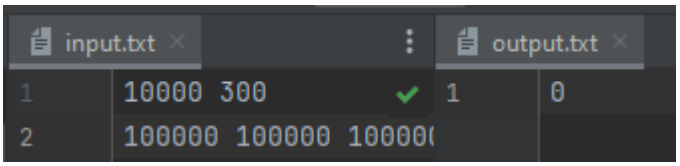
Функция max_count_gold возвращает первый с конца списка dp индекс элемента, в котором содержится единица. Этот индекс и соответствует максимальному весу, который можно собрать и поместить в рюкзак.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.000434899935498833 66	18,13
Пример из задачи	0.000335199991241097 45	18,19
Верхняя граница диапазона значений входных данных из текста задачи	0.002446345673954878 3	184,34

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

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

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

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

- **Постановка задачи.** Найдите максимальное значение арифметического выражения, указав порядок применения его арифметических операций с помощью дополнительных скобок.
- **Формат ввода / входного файла (input.txt).** Единственная строка входных данных содержит строку s длины $2n + 1$ для некоторого n с символами $s_0, s_1, ..., s_{2n}$. Каждый символ в четной позиции s является цифрой (то есть целым числом от 0 до 9), а каждый символ в нечетной позиции является одной из трех операций из $+, -, *$
- **Ограничения на входные данные.** $0 \leq n \leq 14$ (следовательно, строка содержит не более 29 символов).
- **Формат вывода / выходного файла (output.txt).** Выведите максимально возможное значение заданного арифметического выражения среди различных порядков применения арифметических операций.
- Ограничение по времени. 5 сек.
- Пример:

input.txt	output.txt	input.txt	output.txt
1+5	6	5-8+7*4-8+9	200

Здесь $200 = (5 - ((8 + 7) * (4 - (8 + 9))))$.

Листинг кода

```
import time
import tracemalloc


def min_and_max_value(i, j, m, M, operations):
    minimum = float('+inf')
    maximum = float('-inf')
    for k in range(i, j):
        a = eval(M[i][k] + operations[k] + M[k + 1][j])
        b = eval(M[i][k] + operations[k] + m[k + 1][j])
        c = eval(m[i][k] + operations[k] + M[k + 1][j])
        d = eval(m[i][k] + operations[k] + m[k + 1][j])
```

```
            minimum = min(minimum, a, b, c, d)
            maximum = max(maximum, a, b, c, d)
    return minimum, maximum


def get_max_value(digits, operations):
    count_numbers = len(digits)
    m = [[0] * count_numbers for _ in range(count_numbers)]
    M = [[0] * count_numbers for _ in range(count_numbers)]
    for i in range(count_numbers):
        m[i][i] = digits[i]
        M[i][i] = digits[i]
    for s in range(1, count_numbers):
        for i in range(count_numbers - s):
            j = i + s
            m[i][j], M[i][j] = map(str, min_and_max_value(i, j, m, M, operations))
    return M[0][count_numbers - 1]


def main():
    with open('input.txt') as f:
        s = f.readline()
        numbers = []
        operations = []
        for i in range(len(s)):
            if i % 2 == 0:
                numbers.append(s[i])
            else:
                operations.append(s[i])

    with open("output.txt", "w") as f:
        f.write(str(get_max_value(numbers, operations)))


if __name__ == '__main__':
    tracemalloc.start()
    t_start = time.perf_counter()
```

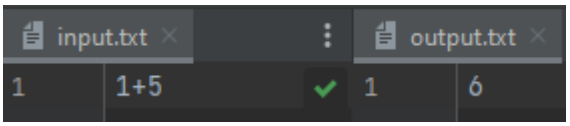
```
main()
t_end = time.perf_counter()
print(f"Время: {t_end - t_start} секунд")
print(f"Память: {tracemalloc.get_traced_memory()}")
tracemalloc.stop()
```

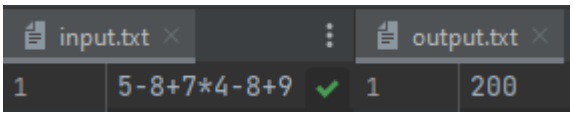
Текстовое объяснение решения

В двумерных списках m и M хранятся соответственно минимальные и максимальные значения подвыражений.

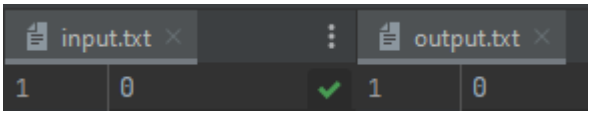
В функции min_and_max_value для каждого подвыражения от i до j для любого разбиения высчитываются четыре результата из комбинации значений до и после точки разделения, из них определяются минимальное и максимальное значения, которые возвращает функция для данного подвыражения.

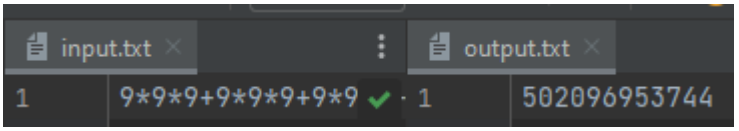
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.000416499795392155 65	17,98
Пример из задачи	0.000564299989491701	24,09

	1	
Пример из задачи	0.00403439998626709	27,28
Верхняя граница диапазона значений входных данных из текста задачи	0.057052599964663386	41,00

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

Задача №21. Игра в дурака [3 баллов]

- **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.

- **Формат входного файла (input.txt).** В первой строке входного файла находятся два натуральных числа N и M, а также символ R, означающий козырную масть. Во второй строке перечислены N карт, находящихся на руках у игрока. В третьей строке перечислены M карт, которые необходимо отбить. Все карты отделены друг от друга одним пробелом.

- **Ограничения на входные данные.** $N \leq 35$, $M \leq 4$, $M \leq N$.

- **Формат выходного файла (output.txt).** В выходной файл выведите «YES» в случае, если отбиться можно, либо «NO», если нельзя.

- Ограничение по времени. 1 сек.

- Ограничение по памяти. 16 мб.

- Примеры:

input.txt	output.txt	input.txt	output.txt
6 2 C KD KC AD 7C AH 9C 6D 6C	YES	4 1 D 9S KC AH 7D 8D	NO

- Проверить можно по ссылке.

Листинг кода

```
import time
import tracemalloc



def game(my_cards, opp_cards, r):
    rank = ['6', '7', '8', '9', 'T', 'J', 'Q', 'K','A']
    suit = ['S', 'C', 'D', 'H']
    for card in opp_cards[r]:
```

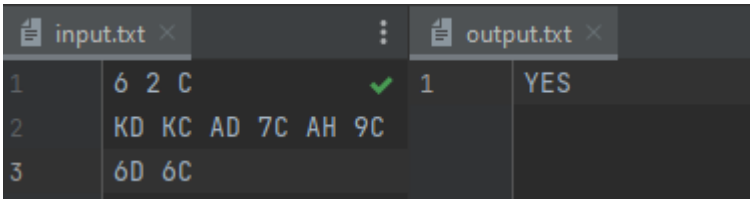
```
            for my_card in my_cards[r]:
                if rank.index(my_card) > rank.index(card):
                    opp_cards[r].remove(card)
                    my_cards[r].remove(my_card)
                    break
            if opp_cards[r]:
                return False
        suit.remove(r)
        for char in suit:
            for card in opp_cards[char]:
                for my_card in my_cards[char]:
                    if rank.index(my_card) > rank.index(card):
                        opp_cards[char].remove(card)
                        my_cards[char].remove(my_card)
                        break
                if opp_cards[char]:
                    for card in opp_cards[char]:
                        if my_cards[r]:
                            opp_cards[char].remove(card)
                            my_cards[r] = my_cards[r][1:]
                        else:
                            return False
                if opp_cards[char]:
                    return False
        return True

    def main():
        with open('input.txt') as f:
            my_cards = {'S': [], 'C': [], 'D': [], 'H': []}
            opp_cards = {'S': [], 'C': [], 'D': [], 'H': []}
            n, m, r = [i for i in f.readline().split()]
            for my_card_m_r in f.readline().split():
                my_cards[my_card_m_r[1]].append(my_card_m_r[0])
            for oppo_card_m_r in f.readline().split():
                opp_cards[oppo_card_m_r[1]].append(oppo_card_m_r[0])
        with open("output.txt", "w") as f:
            f.write(('NO', 'YES')[game(my_cards, opp_cards, r)])
```

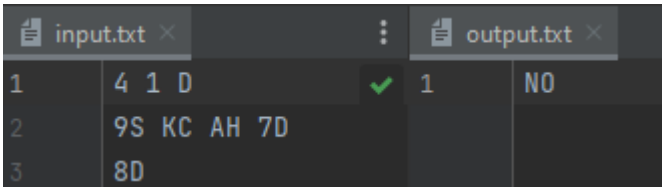
```
if __name__ == '__main__':
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

Текстовое объяснение решения

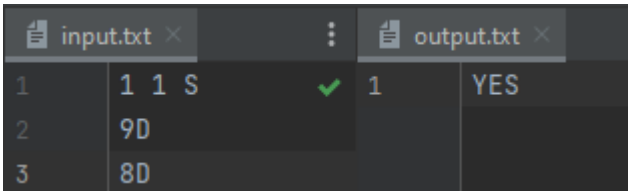
После считывания карт двух игроков в словари по ключу масти. Программа предварительно проходит по козырным картам оппонента и смотрит может ли она их побить. После этого она приступает к обычным картам. В случае, если после прохода по всем картам не получается найти карту, которая может побить карту оппонента, то возвращается ноль. Если же побиты все карты то возвращается единица.

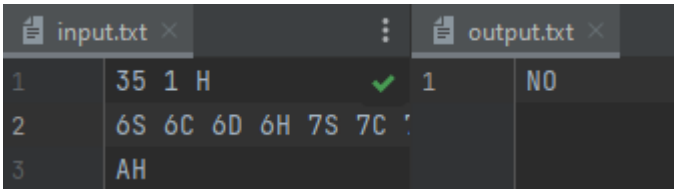
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





Проверка задачи на acmp

Тест	Результат	Время	Память
1	Accepted	0,031	2098 Кб
2	Accepted	0,062	2098 Кб
3	Accepted	0,015	2106 Кб
4	Accepted	0,046	2098 Кб
5	Accepted	0,046	2098 Кб
6	Accepted	0,046	2102 Кб
7	Accepted	0,031	2102 Кб
8	Accepted	0,062	2094 Кб
9	Accepted	0,015	2094 Кб
10	Accepted	0,062	2098 Кб
11	Accepted	0,062	2102 Кб
12	Accepted	0,031	2102 Кб
13	Accepted	0,046	2098 Кб
14	Accepted	0,031	2094 Кб
15	Accepted	0,015	2102 Кб
16	Accepted	0,046	2102 Кб
17	Accepted	0,046	2098 Кб
18	Accepted	0,031	2098 Кб
19	Accepted	0,046	2102 Кб
20	Accepted	0,015	2102 Кб
21	Accepted	0,046	2102 Кб
22	Accepted	0,062	2098 Кб
23	Accepted	0,031	2098 Кб
24	Accepted	0,031	2098 Кб
25	Accepted	0,046	2094 Кб
26	Accepted	0,031	2098 Кб
27	Accepted	0,031	2090 Кб
28	Accepted	0,031	2102 Кб
29	Accepted	0,062	2098 Кб
30	Accepted	0,046	2098 Кб
31	Accepted	0,062	2098 Кб
32	Accepted	0,046	2090 Кб
33	Accepted	0,046	2098 Кб
34	Accepted	0,031	2102 Кб
35	Accepted	0,046	2094 Кб
36	Accepted	0,046	2094 Кб
37	Accepted	0,078	2094 Кб
38	Accepted	0,015	2094 Кб
39	Accepted	0,031	2094 Кб
40	Accepted	0,046	2102 Кб
41	Accepted	0,015	2094 Кб
42	Accepted	0,015	2098 Кб
43	Accepted	0,062	2102 Кб
44	Accepted	0,062	2094 Кб
45	Accepted	0,031	2094 Кб
46	Accepted	0,015	2094 Кб
47	Accepted	0,046	2094 Кб
48	Accepted	0,031	2090 Кб
49	Accepted	0,046	2098 Кб
50	Accepted	0,046	2090 Кб

	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0007152999751269817	28,57
Пример из задачи	0.0004201000556349754 3	28,66
Пример из задачи	0.0004209999460726976 4	28,68
Верхняя граница диапазона значений входных данных из текста задачи	0.0003984000068157911 3	29,14

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

Дополнительные задачи

Задача №1. Максимальная стоимость добычи [0.5 баллов]

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 p_i и w_i – стоимость и вес i-го предмета, соответственно.
- **Ограничения на входные данные.** $1 \leq n \leq 10^3, 0 \leq W \leq 2 \cdot 10^6, 0 \leq p_i \leq 2 \cdot 10^6, 0 \leq w_i \leq 2 \cdot 10^6$ для всех $1 \leq i \leq n$. 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).
- Ограничение по времени. 2 сек.
- Примеры:

input.txt	output.txt
3 50 60 20 100 50 120 30	180.0000

Чтобы получить значение 180, берем первый предмет и третий предмет в сумку.

input.txt	output.txt
1 10 500 30	166.6667

Здесь просто берем одну треть единственного доступного предмета.

Листинг кода

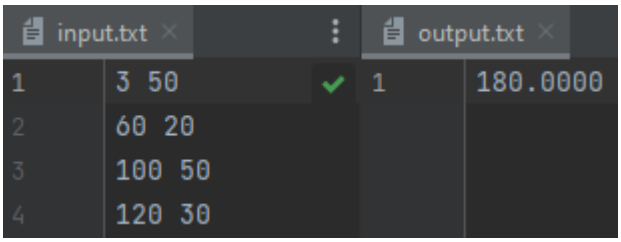
```
import time
import tracemalloc


def max_cost(a, n, w):
    a.sort(key=lambda k: k[0] / k[1] if k[1] != 0 else 0, reverse=True)
    i = 0
    money = 0
```

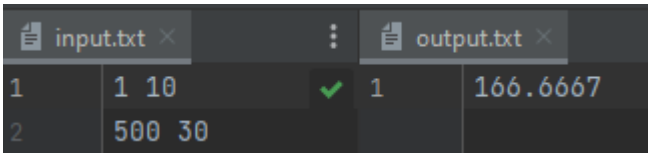
```
    while w > 0 and i < n:
        if a[i][1] <= w:
            money += a[i][0]
            w -= a[i][1]
        else:
            money += a[i][0] / a[i][1] * w
            w = 0
        i += 1
    return money


def main():
    with open("input.txt") as f:
        n, w = map(int, f.readline().split())
        data = [[0, 0] for _ in range(n)]
        for i in range(n):
            data[i][0], data[i][1] = map(int, f.readline().split())

    money = max_cost(data, n, w)
    with open("output.txt", "w") as f:
        f.write(str(format(money, '.4f')))


if __name__ == '__main__':
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

Текстовое объяснение решения

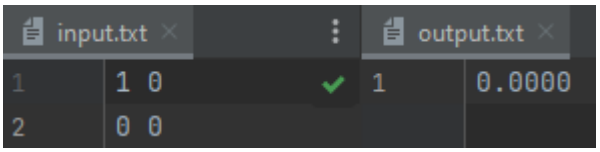
Сортируется список предметов по убыванию удельной стоимости (делится стоимость предмета на его массу). Затем добавляются предметы в сумку, начиная с первого предмета с самой большой удельной стоимостью. Если предмет помещается в сумку целиком, то добавляется к общей сумме

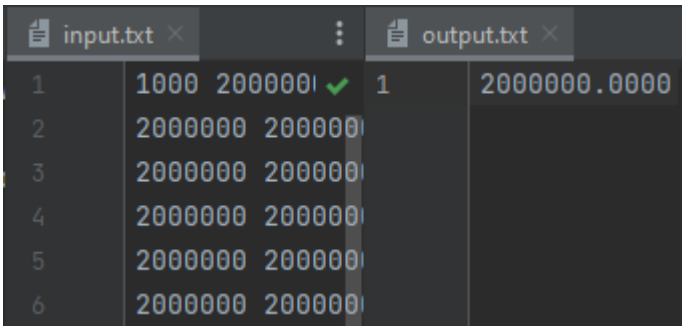
money его стоимость, а из значения w, обозначающего максимальный вес, который может быть в сумке, вычитается его масса. В противном случае к общей сумме прибавляется стоимость вмещающейся части предмета.
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0005098001565784216	18,12
Пример из задачи	0.0004310000222176313 4	18,18
Пример из задачи	0.0007391001563519239	18,13
Верхняя граница диапазона значений	0.0075435624576303415	154,57

входных данных из текста задачи		

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

Задача №13. Сувениры [1.5 баллов]

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.

- **Формат ввода / входного файла (input.txt).** В первой строке дано целое число n. Во второй строке даны целые числа $v_1, v_2, ..., v_n$, разделенные пробелами.
- **Ограничения на входные данные.** $1 \leq n \leq 20, 1 \leq v_i \leq 30$ для всех i.
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если можно разбить $v_1, v_2, ..., v_n$ на три подмножества с одинаковыми суммами и 0 в противном случае.
- Ограничение по времени. 5 сек.
- Примеры:

input.txt	output.txt
4 3 3 3 3	0

input.txt	output.txt
1 40	0

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

Здесь 34 + 67 + 17 = 23 + 59 + 1 + 17 + 18 = 59 + 2 + 57.

input.txt	output.txt
13 1 2 3 4 5 5 7 7 8 10 12 19 25	1

Здесь 1 + 3 + 7 + 25 = 2 + 4 + 5 + 7 + 8 + 10 = 5 + 12 + 19.

Листинг кода

```
import time
import tracemalloc


def is_possible_to_split_souvenirs(souvenirs, n, sum1, sum2, sum3, index, data):
    if index == n:
```

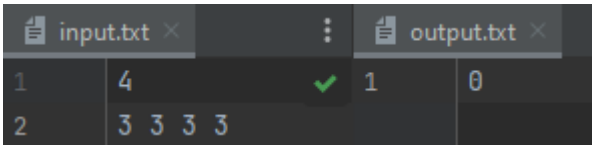
```
        if sum1 == sum2 == sum3:
            return '1'
        else:
            return '0'

    key = (sum1, sum2, sum3, index)
    if key in data:
        return data[key]
    data[key] = max(
            is_possible_to_split_souvenirs(souvenirs, n, sum1 + souvenirs[index],
sum2, sum3, index + 1, data),
                    is_possible_to_split_souvenirs(souvenirs, n, sum1, sum2 +
souvenirs[index], sum3, index + 1, data),
                is_possible_to_split_souvenirs(souvenirs, n, sum1, sum2, sum3 +
souvenirs[index], index + 1, data)
    )
    return data[key]


def main():
    with open('input.txt') as f:
        n = int(f.readline())
        souvenirs = [int(souvenir) for souvenir in f.readline().split()]

    with open("output.txt", "w") as f:
        if sum(souvenirs) % 3 == 0:
            f.write(is_possible_to_split_souvenirs(souvenirs, n, 0, 0, 0, 0, {}))
        else:
            f.write('0')


if __name__ == '__main__':
    tracemalloc.start()
    t_start = time.perf_counter()

    main()
    t_end = time.perf_counter()
```

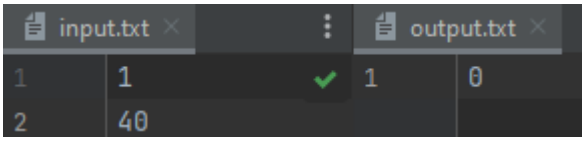
```
print(f'Время: {t_end - t_start} секунд')
print(f'Память: {tracemalloc.get_traced_memory()}')
tracemalloc.stop()
```

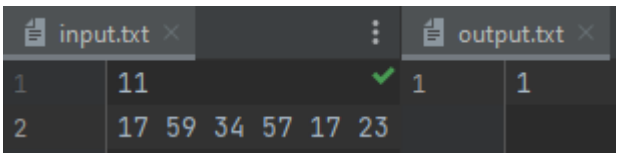
Текстовое объяснение решения

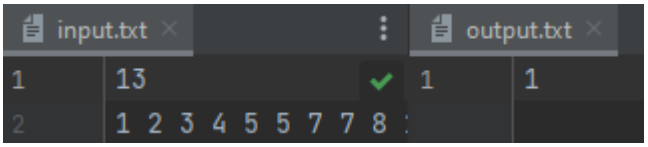
Сначала производится проверка, кратна ли сумма всех сувениров трём. Если это не так, то разделить поровну сувениры между тремя друзьями невозможно. Иначе вызывается функция is_possible_to_split_souvenirs с параметрами: souvenirs – исходный список сувениров; n – количество сувениров; sum1, sum2 и sum3 – суммы в трёх подмножествах сувениров; index – индекс текущего сувенира в списке; data – ассоциативный список, в который записываются промежуточные значения для ускорения работы рекурсии (ключом в словаре dp является кортеж, состоящий из текущих значений сумм в трёх подмножествах сувениров и индекса текущего сувенира). Данная функция вызывается рекурсивно. На каждом вызове добавляется сувенир из списка с индексом index к одному из трёх подмножеств. Таким образом рассматриваются все возможные варианты распределения сувениров по трём подмножествам. Базовым случаем рекурсии является ситуация, когда индекс текущего элемента равен количеству сувениров. Если к этому моменту суммы всех трёх подмножеств сувениров равны, то функция возвращает единицу — это означает, что сувениры можно разделить поровну, иначе функция возвращает ноль. В словарь data записывается максимальное значение, то есть «1», если она, конечно, есть.

Результат работы кода на примерах из текста задачи:

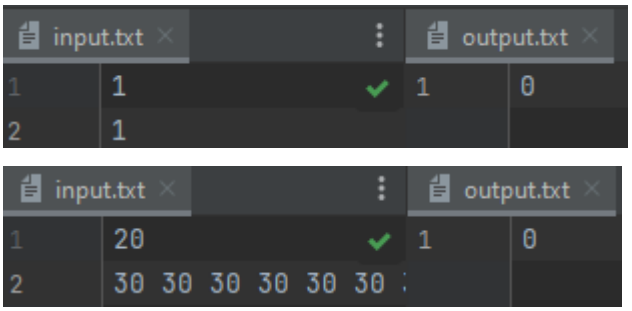








Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0003332998603582382	18,61
Пример из задачи	0.0004022999200969934 5	18,67
Пример из задачи	0.0003579999320209026 3	18,66
Пример из задачи	0.060346600133925676	3494,44
	0.023015799932181835	999,75
Верхняя граница диапазона значений входных данных из текста задачи	0.005190399941056967	239,60

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

Задача №15. Удаление скобок [2 баллов]

- **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.
- **Формат ввода / входного файла (input.txt).** Во входном файле записана строка, состоящая из s символов: круглых, квадратных и фигурных скобок $(), [], \{\}$. Длина строки не превосходит 100 символов.
- **Ограничения на входные данные.** $1 \leq s \leq 100$.
- **Формат вывода / выходного файла (output.txt).** Выведите строку максимальной длины, являющейся правильной скобочной последовательностью, которую можно получить из исходной строки удалением некоторых символов.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
([)]	[]

Листинг кода

```
import time
import tracemalloc


def get_answer(dp, pred, s, left, right, res):
    if right == left:
        pass
    elif dp[left][right] == 0:
        for k in range(left, right + 1):
            res += s[k]
    elif pred[left][right] == -1:
        res += s[left]
        res = get_answer(dp, pred, s, left + 1, right - 1, res)
        res += s[right]
    else:
        res = get_answer(dp, pred, s, left, pred[left][right], res)
        res = get_answer(dp, pred, s, pred[left][right] + 1, right, res)
    return res
```

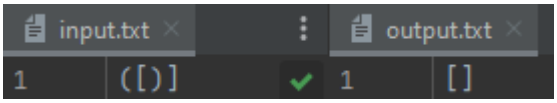
```
def remove_brackets(s):
    maxsize = 101
    n = len(s)
    dp = [[0] * n for _ in range(n)]
    pred = [[0] * n for _ in range(n)]
    for i in range(n):
        dp[i][i] = 1

    for right in range(1, n):
        for left in range(right - 1, -1, -1):
            if (s[left] == '(' and s[right] == ')') or (s[left] == '[' and s[right] == ']') or
(s[left] == '{' and s[right] == '}'):
                cur = dp[left + 1][right - 1]
            else:
                cur = maxsize
            pred[left][right] = -1
            for k in range(left, right):
                if dp[left][k] + dp[k + 1][right] < cur:
                    cur = dp[left][k] + dp[k + 1][right]
                    pred[left][right] = k

            dp[left][right] = cur
    return get_answer(dp, pred, s, 0, n - 1, '')


def main():
    with open('input.txt') as f:
        s = f.readline()

    with open("output.txt", "w") as f:
        f.write(remove_brackets(s))


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
```

```
main()
t_end = time.perf_counter()
print(f"Время: {t_end - t_start} секунд")
print(f"Память: {tracemalloc.get_traced_memory()}")
tracemalloc.stop()
```

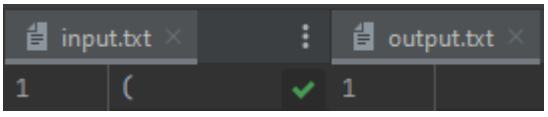
Текстовое объяснение решения

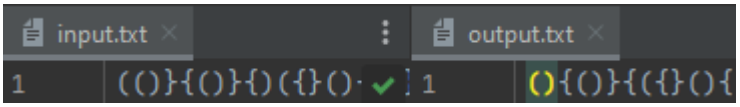
Заполняются таблица dp, в которой хранятся числа, равные минимальному количеству удалений, которые нужно совершить на подстроке от left до right включительно, и таблица pred, в которую заносится индекс элемента, по которому можно разбить подстроку на две подстроки, суммарное количество удалений в которых будет минимально для данной подстроки. 16 Ответ восстанавливается с помощью рекурсивной функции get_answer, в которую передаются таблицы dp и pred, исходная строка s со всеми скобками, левая граница left и правая граница right, строка res с правильной скобочной последовательностью. Если левая и правая границы совпадают, то возвращается строка res без изменений. Если значение в таблице dp для данной подстроки равно 0, то к строке res дописываются по очереди элементы подстроки. Если значение в таблице pred для данной подстроки равно -1 (означает, что у данной подстроки нет элемента, по которому её можно разбить для получения более оптимального решения), то к res дописываются первый элемент подстроки, вызывается функция get_answer, и затем к res добавляется последний элемент подстроки. В противном случае рекурсивно вызывается функция get_answer для подстрок, разделённых тем самым элементом.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.00029930006712675095	17,92
Пример из задачи	0.0003861999139189 72	17,92
Верхняя граница диапазона значений входных данных из текста задачи	0.025563600221648812	197,14

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

Задача №19. Произведение матриц [3 баллов]

- **Постановка задачи.** 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.
- **Формат ввода / входного файла (input.txt).** В первой строке входных данных содержится целое число n - количество матриц. В n следующих строк содержится по два целых числа a_i и b_i - количество строк и столбцов в i-той матрице соответственно. Гарантируется, что $b_i = a_{i+1}$ для любого $1 \leq i \leq n-1$.
- **Ограничения на входные данные.** $1 \leq n \leq 400$, $1 \leq a_i, b_i \leq 100$ для всех $1 \leq i \leq n$.
- **Формат вывода / выходного файла (output.txt).** Выведите оптимальную расстановку скобок. Если таких расстановок несколько, выведите любую.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 64 мб.
- Пример:

input.txt	output.txt
3 10 50 50 90 90 20	((AA)A)

- В данном примере возможно всего две расстановки скобок: $((AA)A)$ и $(A(AA))$. При первой количество операций будет равно $10 \cdot 50 \cdot 90 + 10 \cdot 90 \cdot 20 = 63000$, а при второй - $10 \cdot 50 \cdot 20 + 50 \cdot 90 \cdot 20 = 100000$.

Листинг кода

```
import time
import tracemalloc


def get_answer(i, j, brackets, res):
    if i == j:
        res += 'A'
        return res
    res += '('
```

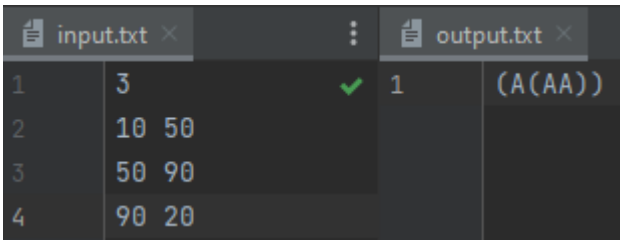
```
    res = get_answer(i, brackets[i][j], brackets, res)
    res = get_answer(brackets[i][j] + 1, j, brackets, res)
    res += ')'
    return res


def matrix_mult(A, n):
    dp = [[0] * (n + 1) for _ in range(n + 1)]
    brackets = [[0] * (n + 1) for _ in range(n + 1)]

    for right in range(2, n + 1):
        for left in range(right - 1, 0, -1):
            cur = 10 ** 9
            for k in range(left, right):
                cost = dp[left][k] + dp[k + 1][right] + A[left - 1] * A[k] * A[right]
                if cost < cur:
                    cur = cost
                    brackets[left][right] = k
        dp[left][right] = cur
    return get_answer(1, n, brackets, '')


def main():
    with open('input.txt') as f:
        n = int(f.readline())
        A = [tuple(map(int, x.split())) for x in f.readlines()]

    B = [0] * (n + 1)
    for i in range(n):
        B[i] = A[i][0]
    B[n] = A[n - 1][1]
    with open("output.txt", "w") as f:
        f.write(matrix_mult(B, n))


if __name__ == '__main__':
    tracemalloc.start()
```

```
t_start = time.perf_counter()
main()
t_end = time.perf_counter()
print(f"Время: {t_end - t_start} секунд")
print(f"Память: {tracemalloc.get_traced_memory()}")
tracemalloc.stop()
```

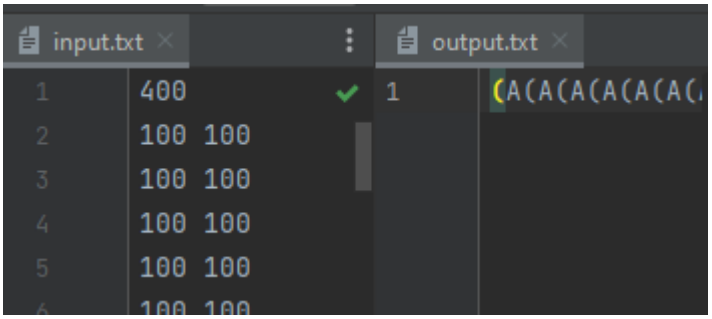
Текстовое объяснение решения

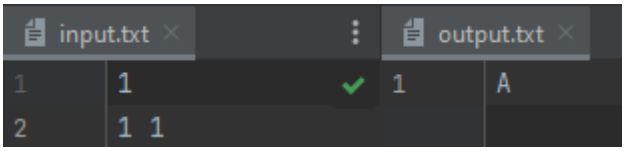
Заполняется таблица dp, в которой хранятся числа, равные минимальному количеству операций, которые нужно совершить при перемножении матриц, и таблица brackets, в которую заносится индекс матрицы, который определяет оптимальное разбиение на каждом шаге. Ответ восстанавливается с помощью рекурсивной функции get_answer. Если индекс разделения совпадает с индексом начальной или конечной матрицы в подцепочке, то к строке res прибавляется название матрицы (оно у всех одинаковое – «A»). В противном случае к res прибавляется открывающая скобка, рекурсивно вызывается функция get_answer и результат присваивается res, затем к res добавляется закрывающая скобка.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ

Нижняя граница диапазона значений входных данных из текста задачи	0.000408499967306852 34	18,25
Пример из задачи	0.000452800188213586 8	18,38
Верхняя граница диапазона значений входных данных из текста задачи	14.161617599893361	3326,88

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

Вывод

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