

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 №4

Выполнил:
Волжева М.И.
К3141

Проверила:
Артамонова В.Е.

Санкт-Петербург
2023 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №2. Заправки (0.5 балла)	3
Задача №7. Проблема сапожника (0.5 балла)	6
Задача №11. Максимальное количество золота (1 балл)	9
Задача №16. Продавец (2 балла)	12
Задача №17. Ход конем (2.5 балла)	15
Дополнительные задачи	18
Задача №1. Максимальная стоимость добычи (0.5 балла)	18
Задача №3. Максимальный доход от рекламы (0.5 балла)	21
Задача №5. Максимальное количество призов (0.5 балла)	23
Задача №6. Максимальная зарплата (0.5 балла)	26
Задача №8. Расписание лекций(1 балл)	29
Задача №9. Распечатка (1 балл)	32
Задача №10. Яблоки (1 балл)	35
Задача №12. Последовательность (1 балл)	38
Задача №13. Сувениры (1.5 балла)	41
Задача №15. Удаление скобок (2 балла)	44
Вывод	47

Задачи по варианту

Задача №2. Заправки (0.5 балла)

- Текст задачи:

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

- Формат входного файла (input.txt).

В первой строке содержится d - целое число. Во второй строке - целое число m . В третьей находится количество заправок на пути - n . И, наконец, в последней строке – целые числа через пробел - остановки $stop_1, stop_2, \dots, stop_n$.

- Ограничения на входные данные.

$1 \leq d \leq 10^5, 1 \leq m \leq 400, 1 \leq n \leq 300, 1 < stop_1 < stop_2 < \dots < stop_n < d$

- Формат 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 $stop_1, stop_2, \dots, stop_n$ 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. 2сек.

- Пример:

Input.txt	950 400 4 200 375 550 750	10 3 4 1 2 5 9	200 250 2 100 150
Output.txt	2	-1	0

Листинг кода:

```
import time
import os, psutil

def decision(my_distance, stops):
    stops_new = []
    stops_new.append(stops[0])
```

```

for i in range(1, len(stops)):
    stops_new.append(stops[i] - stops[i-1])
sum = 0
count = 0
for i in stops_new:
    sum += i
    if sum > my_distance:
        count += 1
        sum = i
    if i > my_distance:
        return -1
return count







process = psutil.Process(os.getpid())
t_start = time.perf_counter()
f = open("2_input.txt")
m = open("2_output.txt", "w")
all_distance = int(f.readline())
my_distance = int(f.readline())
count = int(f.readline())
stops = list(map(int, f.readline().split()))
stops.append(all_distance)
count_necessary_stops = decision(my_distance, stops)
m.write(str(count_necessary_stops))
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

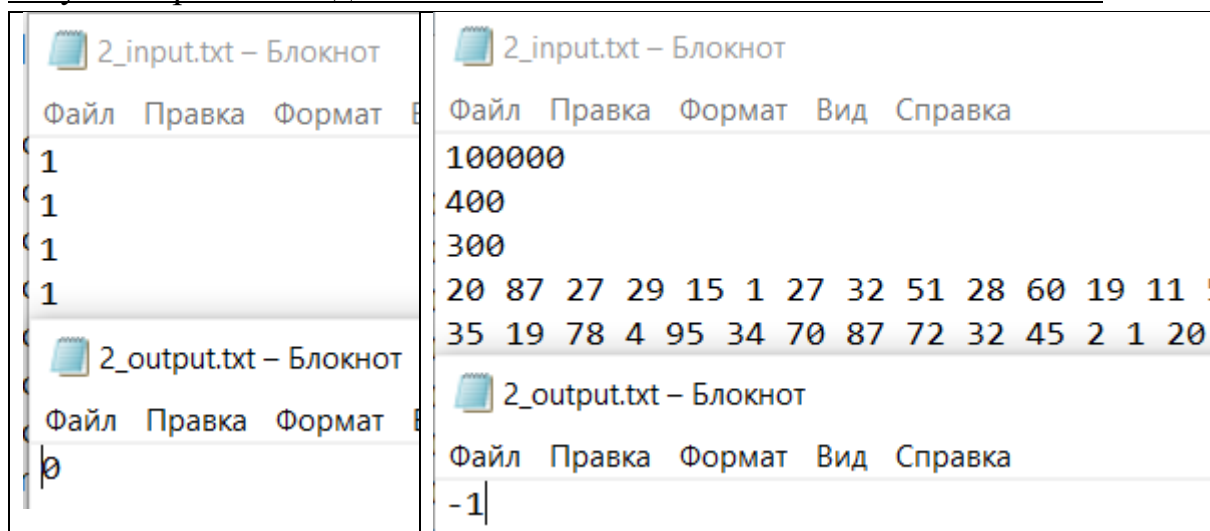
Текстовое объяснение решения:

На вход функции подаётся массив с набором расстояний от начала до заправок. Первым шагом нашего решения является перевод этого набора данных в другой – массив расстояний между соседними заправками. Далее мы проходимся по новому набору данных и рассматриваем возможность пропуска каждой заправки, при невозможности добавляем ее в список заправок.

Результат работы кода на примерах из текста задачи:

 2_input.txt – Блокнот Файл Правка Формат 950 400 4 200 375 550 750	 2_input.txt – Блокнот Файл Правка Формат 10 3 4 1 2 5 9	 2_input.txt – Блокнот Файл Правка Формат 200 250 2 100 150
 2_output.txt – Блокнот Файл Правка Формат 2	 2_output.txt – Блокнот Файл Правка Формат -1	 2_output.txt – Блокнот Файл Правка Формат 0

Результат работы кода на максимальных и минимальных значениях:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.008535299999999968	14.140625
Пример №1	0.0011470999999999842	14.14453125
Пример №2	0.00146030000000000255	14.140625
Пример №3	0.0009000999999999987	14.14453125
Максимальное значение	0.0017291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №7. Проблема сапожника (0.5 балла)

- Текст задачи:

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

- Формат входного файла (input.txt).

В первой строке вводятся натуральные числа K и n . Затем во второй строке идет n натуральных чисел t_1, \dots, t_n - количество минут, которые требуются, чтобы починить i -й сапог.

- Ограничения на входные данные.

$1 \leq K \leq 1000$, $1 \leq n \leq 500$, $1 \leq t_i \leq 100$ для всех $1 \leq i \leq n$

- Формат выходного файла (output.txt).

Выведите одно число – максимальное количество сапог, которые можно починить за один рабочий день.

- Ограничение по времени. 2сек.

- Ограничение по памяти. 256 мб.

- Пример:

Input.txt	10 3 6 2 8	3 2 10 20
Output.txt	2	0

Листинг кода:

```
import time
import os, psutil

def decision(minutes, all_minutes):
    minutes = sorted(minutes)
    sum_minutes = 0
    counts = 0
    for i in minutes:
        sum_minutes += i
        if sum_minutes <= all_minutes:
            counts += 1
        else:
            break
    return counts

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("7 input.txt")
```

```

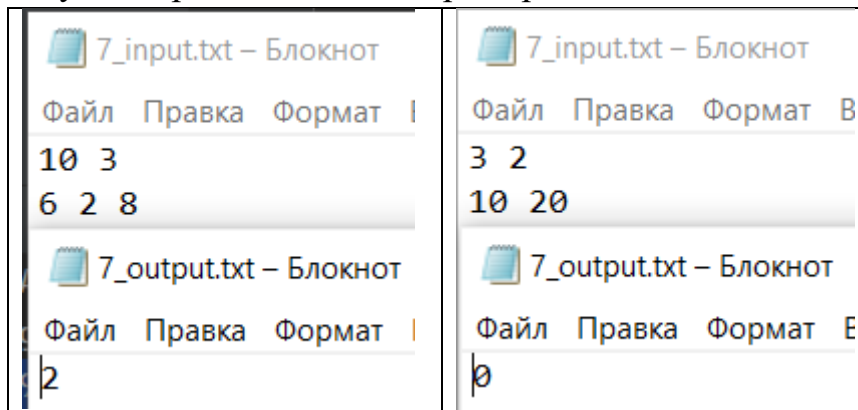
m = open("7_output.txt", "w")
all_minutes, shoes = map(int, f.readline().split())
minutes = list(map(int, f.readline().split()))
minutes = sorted(minutes)
count_shoes = decision(minutes, all_minutes)
m.write(str(count_shoes))
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

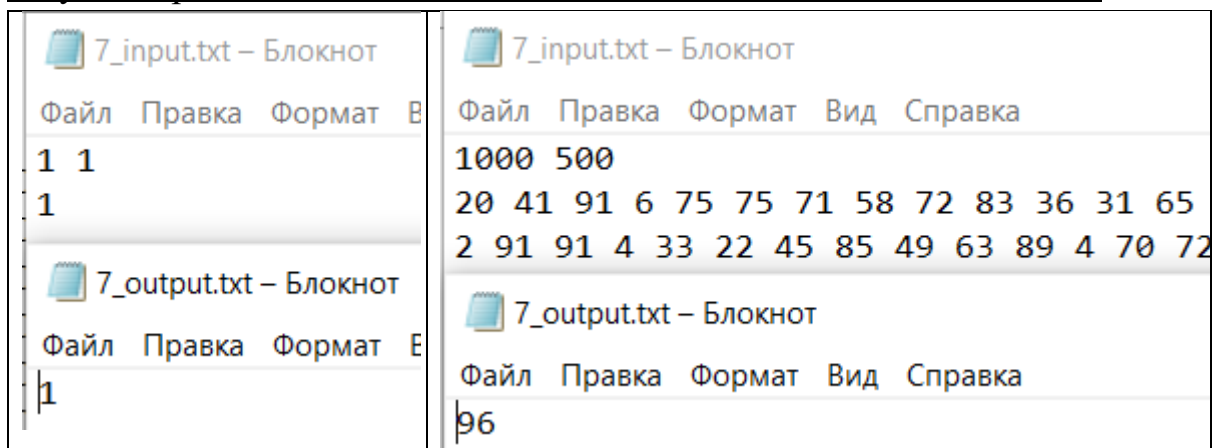
Текстовое объяснение решения:

В функцию подаётся массив чисел и максимальное значение, первым шагом решения является сортировка массива этих чисел по возрастанию. Далее мы последовательно проходимся по этим данным и суммируем до того момента, как сумма не станет больше максимального значения или не закончится массив. Исходя из получившегося конца нашего суммирования функция подаёт на выход – максимально возможное количество чисел для суммирования.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.004863699999999971	14.05859375
Пример №1	0.0016550000000000176	14.05859375
Пример №2	0.0015656000000000003	14.05859375
Максимальное значение	0.0007737000000000022	13.92578125

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №11. Максимальное количество золота (1 балл)

- Текст задачи:

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). Даны n золотых слитков, найдите максимальный вес золота, который поместится в сумку вместимостью W .

- Формат 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, \dots, w_{n-1} , определяющие вес золотых слитков.

- Ограничения на входные данные.

$1 \leq W \leq 10^4, 1 \leq n \leq 300, 0 \leq w_0, \dots, w_{n-1} \leq 10^5$

- Формат выходного файла (output.txt).

Выведите максимальный вес золота, который поместится в сумку вместимости W .

- Ограничение по времени. 5 сек.

- Пример:

Input.txt	10 3 1 4 8
Output.txt	9

Листинг кода:

```
import time
import os, psutil

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
capacity, quantity = map(int, f.readline().split())
weights = list(map(int, f.readline().split()))

weights.sort()
dp = []
for i in range(quantity + 1):
    dp.append([0] * (capacity + 1))

for i in range(1, quantity + 1):
    for j in range(1, capacity + 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])

m.write(str(dp[quantity][capacity]) + "\n")

t_end = time.perf_counter()
process.cpu_percent(interval=1)
```

```

        dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weights[i - 1]] +
weights[i - 1])

print(dp)
m.write(str(dp[quantity][capacity]))

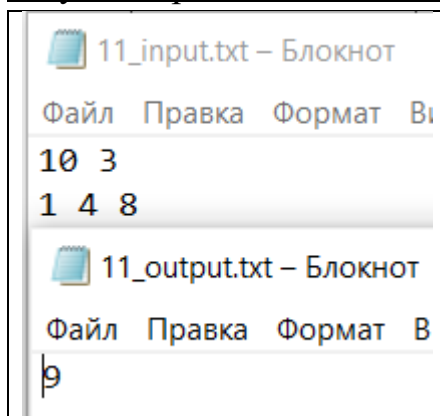
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

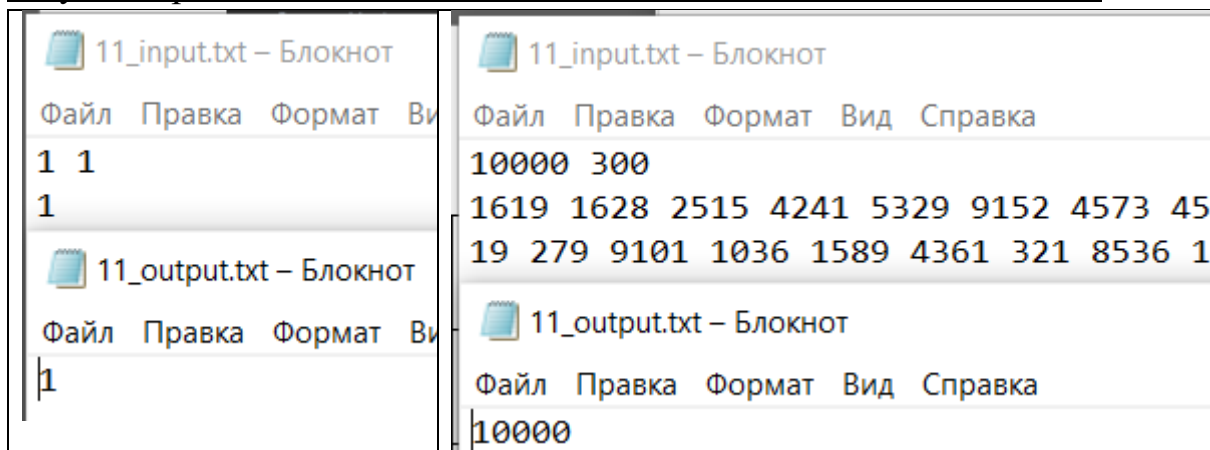
Текстовое объяснение решения:

Мы создаем матрицу из (quantity + 1) строк и (capacity + 1) столбцов и заполняем ее нулями. Затем постепенно проходимся по ней, начиная с начала. Каждая строка – это попытка добавить очередной вес, каждый столбец – максимальный вес (учитывая возможность добавить и максимальное значение для каждого столбца). При добавление веса у нас 2 ситуации, он больше, чем допустимое в столбце, следовательно мы не можем его добавить, переписываем значение слева. Во втором случае мы сравниваем то больше предыдущее значение слева, или добавление к предыдущему слева, сдвинутому на максимальный вес столбца, плюс рассматриваемый вес.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.004863699999999971	14.05859375
Пример №1	0.00156560000000000003	14.05859375
Максимальное значение	0.1309237	14.15234375

Вывод по задаче: Мы научились решать задачи с использованием динамического программирования.

Задача №16. Продавец (2 балла)

- Текст задачи:

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

- Формат входного файла (input.txt).

Первая строка входного файла содержит натуральное число n – количество городов. Следующие n строк содержат по n чисел – длины путей между городами. В i -й строке j -е число – $a_{i,j}$ – это расстояние между городами i и j .

- Ограничения на входные данные.

$1 \leq n \leq 13$, $0 \leq a_{i,j} \leq 10^6$, $a_{i,j} = a_{j,i}$, $a_{i,i} = 0$.

- Формат выходного файла (output.txt).

В первой строке выходного файла выведите длину кратчайшего пути. Во второй строке выведите n чисел – порядок, в котором нужно посетить города.

- Ограничение по времени. 1сек.

- Ограничение по памяти. 256 мб.

- Пример:

Input.txt	5 0 183 163 173 181 183 0 165 172 171 163 165 0 189 302 173 172 189 0 167 181 171 302 167 0
Output.txt	666 4 5 2 3 1

Листинг кода:

```
import time
import os, psutil
import math

def find_right_way(arr, n):
    visited_cities = [False] * n
    lengths = [math.inf] * n
    result = [None] * n
    current = 0
    lengths[current] = 0
    for i in range(0, n):
        min_path_length = float('inf')
        for j in range(0, n):
            if not visited_cities[j] and lengths[j] < min_path_length:
                min_path_length = lengths[j]
```

```

        current = j
        visited_cities[current] = True
        result[current] = i + 1
        for j in range(0, n):
            if not visited_cities[j] and arr[current][j] < lengths[j]:
                lengths[j] = arr[current][j]
        return sum(lengths), reversed(result)

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")

n = int(f.readline())
roads = [list(map(int, f.readline().split())) for i in range(n)]

length, order = find_right_way(roads, n)
m.write(str(length) + "\n")
for i in order:
    m.write(str(i) + " ")

f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

Мы проходимся по всем городам и на каждом шаге смотрим до какого города расстояние ближе всего. В него и идем, а далее рассматриваем оставшиеся города и так до конца. Во время всего этого мы отмечаем, в каких городах уже были и в каком порядке их прошли.

Результат работы кода на примерах из текста задачи:

output.txt	input.txt
1 666	1 5
2 4 5 2 3 1	2 0 183 163 173 181
	3 183 0 165 172 171
	4 163 165 0 189 302
	5 173 172 189 0 167
	6 181 171 302 167 0

Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.008535299999999968	14.140625
Пример №1	0.0006128000095486641	14.671875

Максимальное значение	0.0017291999999999863	14.171875
-----------------------	-----------------------	-----------

Вывод по задаче: Мы научились решать задачи с использованием динамического программирования

Задача №17. Ход конем (2.5 балла)

Текст задачи:

Шахматная ассоциация решила оснастить всех своих сотрудников такими телефонными номерами, которые бы набирались на кнопочном телефоне ходом коня. Например, ходом коня набирается телефон 340-49-27. При этом телефонный номер не может начинаться ни с цифры 0, ни с цифры 8. Напишите программу, определяющую количество телефонных номеров длины N, набираемых ходом коня. Поскольку таких номеров может быть очень много, выведите ответ по модулю 109

- Формат ввода или входного файла (input.txt). Во входном файле записано одно целое число N. ($1 \leq N \leq 1000$.)
- Формат выхода или выходного файла (output.txt). Выведите в выходной файл искомое количество телефонных номеров по модулю 109.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 256 мб.
- Пример:

Input.txt	1	2
Output.txt	8	16

Листинг кода:

```
import time
import os, psutil

def horse(n):
    button = [0] * 10
    if n == 1:
        return 8
    else:
        button = [2, 1, 2, 1, 2, 0, 2, 2, 2, 2]
        arr = [0] * 10
        for i in range(n - 2):
            arr[0] = button[4] + button[6]
            arr[1] = button[6] + button[8]
            arr[2] = button[7] + button[9]
            arr[3] = button[4] + button[8]
            arr[4] = button[0] + button[3] + button[9]
            arr[5] = 0
            arr[6] = button[0] + button[1] + button[7]
            arr[7] = button[2] + button[6]
            arr[8] = button[1] + button[3]
            arr[9] = button[2] + button[4]
        button = arr
```

```

        arr = [0] * 10

    return sum(button) % 10**9

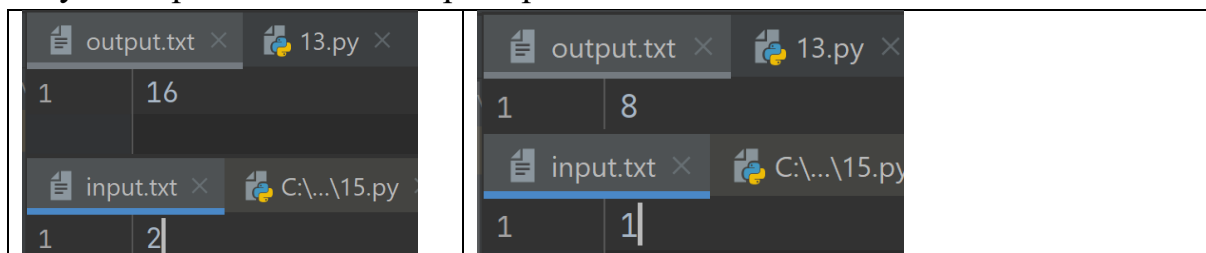
t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
number = int(f.readline())
m.write(str(horse(number)))
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

Для каждой клавиши мы рассчитали количество возможных ходов, которые могут быть сделаны из нее, а далее мы проходимся по всем клавишам и считаем значения по ранее рассчитанным формулам.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения	Затраты памяти
Пример 1 из задачи	0.001938200000000001 second	13.48828125 mb
Пример 2 из задачи	0.0006783999997423962 second	14.53125 mb

Вывод по задаче: Мы научились решать задачи с использованием динамического программирования.

Дополнительные задачи

Задача №1. Максимальная стоимость добычи (0.5 балла)

- Текст задачи:

Вор находит гораздо больше добычи, чем может поместиться в его сумку. 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 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 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. 2сек.

- Пример:

Input.txt	3 50 60 20 100 50 120 30	1 10 500 30
Output.txt	180.0000	166.6667

Листинг кода:

```
import time
import os, psutil
```

```

def knapsack(capacity, values, weights):
    items = []
    for i in range(len(values)):
        itemInfo = {
            'vpw': values[i] / weights[i],
            'weight': weights[i]
        }
        if len(items) == 0:
            items.append(itemInfo)
        else:
            k = 0
            while k < len(items) and items[k]['vpw'] > itemInfo['vpw']:
                k += 1
            items.insert(k, itemInfo)
    total = 0
    capacity_left = capacity
    for item in items:
        if capacity_left - item['weight'] >= 0:
            total += item['weight'] * item['vpw']
            capacity_left -= item['weight']
        elif capacity_left > 0:
            total += item['vpw'] * capacity_left
            capacity_left = 0
    return total

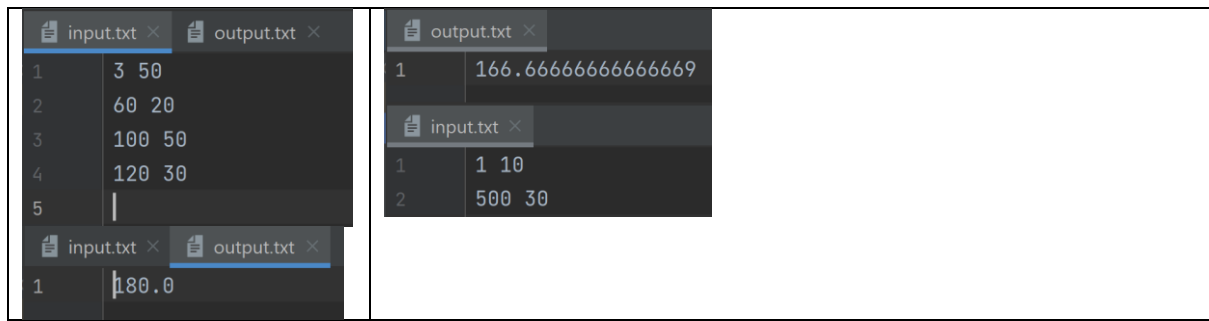
process = psutil.Process(os.getpid())
t_start = time.perf_counter()
f = open("input.txt")
m = open("output.txt", "w")
main_info = list(map(int, f.readline().split()))
capacity = main_info[1]
count = main_info[0]
values = []
weights = []
for i in range(count):
    info = list(map(int, f.readline().split()))
    values.append(info[0])
    weights.append(info[1])
m.write(str(knapsack(capacity, values, weights)))
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

Сначала мы читаем данные и создаем словарь, в котором для каждого товара написан его вес(weight) и коэффициент полезности(vpw). После этого сортируем по убыванию коэффициента полезности и добавляем в сумку элементы, начиная с начала словаря, каждый раз проверяю влезет ли он полностью, при отрицательном ответе на последний вопрос, добавляем лишь его часть, которая влезет.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.008535299999999968	14.140625
Пример №1	0.004355299985036254	14.234375
Пример №2	0.0014603000000000255	14.140625
Максимальное значение	0.0017291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов (задача о дробном рюкзаке).

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

- Текст задачи:

У вас есть n объявлений для размещения на популярной интернет-странице. Для каждого объявления вы знаете, сколько рекламодатель готов платить за один клик по этому объявлению. Вы настроили n слотов на своей странице и оценили ожидаемое количество кликов в день для каждого слота. Теперь ваша цель - распределить рекламу по слотам, чтобы максимизировать общий доход. Даны две последовательности a_1, a_2, \dots, a_n (a_i - прибыль за клик по i -му объявлению) и b_1, b_2, \dots, b_n (b_i - среднее количество кликов в день i -го слота), нужно разбить их на n пар (a_i, b_j) так, чтобы сумма их произведений была максимальной.

- Формат входного файла (input.txt).

В первой строке содержится целое число n , во второй - последовательность целых чисел a_1, a_2, \dots, a_n , в третьей - последовательность целых чисел b_1, b_2, \dots, b_n .

- Ограничения на входные данные.

$1 \leq n \leq 10^3$, $-10^5 \leq a_i, b_i \leq 10^5$, для всех $1 \leq i \leq n$.

- Формат выходного файла (output.txt).

Выведите максимальное значение $\sum_{i=1}^n a_i * c_i$, где c_1, c_2, \dots, c_n является перестановкой b_1, b_2, \dots, b_n .

- Ограничение по времени. 2сек.

- Пример:

Input.txt	1 23 39	3 1 3 -5 -2 4 1
Output.txt	897	23

Листинг кода:

```
import time
import os, psutil

def decision(count, values, weights):
    total = 0
    values.sort()
    weights.sort()
    for i in range(count):
        total += values[i] * weights[i]
    return total
```

```

process = psutil.Process(os.getpid())
t_start = time.perf_counter()
f = open("input.txt")
m = open("output.txt", "w")
count = int(f.readline())
values = list(map(int, f.readline().split()))
weights = list(map(int, f.readline().split()))
m.write(str(decision(count, values, weights)))
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

Читаем данные из файла и записываем их в два массива, каждый сортируем по возрастанию и последовательно соответственно перемножаем. На место, где больше всего кликов должно быть самое высокооплачиваемое заявление.

Результат работы кода на примерах из текста задачи:

<div>output.txt × 3.py ×</div> <div>1 397</div> <div>input.txt ×</div> <div>1 1</div> <div>2 23</div> <div>3 39</div>	<div>output.txt × 3.py ×</div> <div>1 23</div> <div>input.txt ×</div> <div>1 3</div> <div>2 1 3 -5</div> <div>3 -2 4 1</div>
---	--

Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.008535299999999968	14.140625
Пример №1	0.0028208000000336021	14.453125
Пример №2	0.005318599985912442	14.4453125
Максимальное значение	0.0017291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №5. Максимальное количество призов (0.5 балла)

- Текст задачи:

Необходимо представить заданное натуральное число n в виде суммы как можно большего числа попарно различных натуральных чисел. То есть найти максимальное k такое, что n можно записать как $a_1 + a_2 + \dots + a_k$, где a_1, \dots, a_k - натуральные числа и $a_i \neq a_j$ для всех $1 \leq i < j \leq k$.

- Формат входного файла (input.txt).

Входные данные состоят из одного целого числа n .

- Ограничения на входные данные.

$$1 \leq n \leq 10^9$$

- Формат выходного файла (output.txt).

В первой строке выведите максимальное число k такое, что n можно представить в виде суммы k попарно различных натуральных чисел. Во второй строке выведите эти k попарно различных натуральных чисел, которые в сумме дают n (если таких представлений много, выведите любое из них).

- Ограничение по времени. 2сек.

- Пример:

Input.txt	6	8	2
Output.txt	3 1 2 3	3 1 2 5	1 2

Листинг кода:

```
import time
import os, psutil
import math

def decision(summa):
    numbers = [1]
    while sum(numbers) <= summa:
        numbers.append(numbers[-1] + 1)

    numbers.pop()
    ostatok = int(summa - sum(numbers))
    numbers[-1] += ostatok
    return numbers

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
summa = int(f.readline())
```

```

our_decision = decision(summa)
m.write(str(len(our_decision)) + "\n")
for i in range(len(our_decision)):
    m.write(str(our_decision[i]) + " ")
f.close()
m.close()

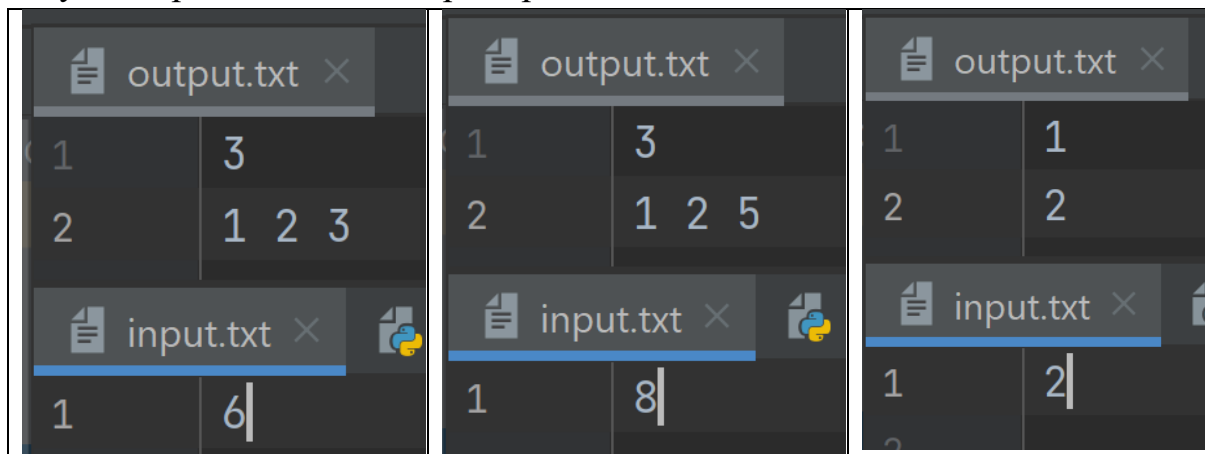
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

В функцию на вход подается число, которое нужно представить как сумму с наибольшим числом положительных неповторяющихся слагаемых. Очевидно, что нужно брать самое маленькое число, а затем большее на единицу. Мы так и будем делать и будем сохранять сумму(тем самым используя динамическое программирование). Если сумма, после прибавления очередного числа станет больше допустимой, то мы вычислим остаток и прибавим его к последнему числу

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.008535299999999968	14.140625
Пример №1	0.0011470999999999842	14.14453125
Пример №2	0.0014603000000000255	14.140625
Пример №3	0.0009000999999999987	14.14453125
Максимальное значение	0.0017291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №6. Максимальная зарплата (0.5 балла)

- Текст задачи:
Составить наибольшее число из набора целых чисел.
- Формат входного файла (input.txt).
Первая строка входных данных содержит целое число n . Во второй строке даны целые числа a_1, a_2, \dots, a_n .
- Ограничения на входные данные.
 $1 \leq n \leq 10^2, 1 \leq a_i \leq 10^3$ для всех $1 \leq i \leq n$.
- Формат выходного файла (output.txt).
Выведите наибольшее число, которое можно составить из a_1, a_2, \dots, a_n .
- Ограничение по времени. 2сек.
- Пример:

Input.txt	2 21 2	3 23 39 92
Output.txt	221	923923

Листинг кода:

```
import time
import os, psutil
import math

def bubble_sort(n, arr):
    for i in range(0, n - 1):
        for j in range(0, n - i - 1):
            x = arr[j] + arr[j + 1]
            y = arr[j + 1] + arr[j]
            if x < y:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    res = ''
    for i in range(0, n):
        res += str(int(arr[i]))
    return res

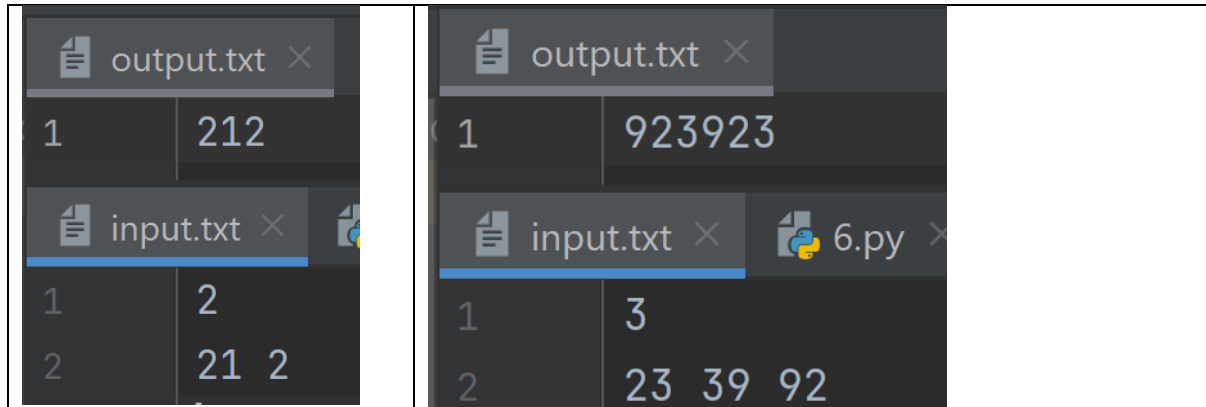
t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
number = int(f.readline())
source_array = f.readline().split(' ')
print(bubble_sort(number, source_array))
m.write(bubble_sort(number, source_array))
f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss / (1024 * 1024), "mb")
```

Текстовое объяснение решения:

Мы сортируем с помощью пузырьковой сортировки. Мы сравниваем два элемента, причем если они начинаются с одной цифры, то сравнение происходит по цифрам, расположенным далее.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.008535299999999968	14.140625
Пример №1	0.002148900064639747	14.44140625
Пример №2	0.0004354999400675297	14. 234375
Максимальное значение	0.0017291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №8. Расписание лекций(1 балл)

- Текст задачи:

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 $[s_i, f_i)$ - время начала и конца лекции. Лекция считается открытым интервалом, то есть какая-то лекция может начинаться в момент окончания другой, без перерыва. Необходимо выбрать из этих заявок такое подмножество, чтобы суммарно выполнить максимальное количество заявок. Учтите, что одновременно.

- Формат входного файла (input.txt).

В первой строке вводится натуральное число N - общее количество заявок на лекции. Затем вводится N строк с описаниями заявок - по два числа в каждом s_i и f_i для каждой лекции i . Гарантируется, что $s_i < f_i$. Время начала и окончания лекции - натуральные числа, не превышают 1440 (в минутах с начала суток).

- Ограничения на входные данные.

$$1 \leq N \leq 1000, 1 \leq s_i, f_i \leq 1440$$

- Формат выходного файла (output.txt).

Выведите одно число— максимальное количество заявок на проведение лекций, которые можно выполнить.

- Ограничение по времени. 2сек.

- Ограничение по памяти. 256мб.

- Пример:

Input.txt	1 5 10	3 1 5 2 3 3 4
Output.txt	1	

Листинг кода:

```

import time
import os, psutil
from operator import itemgetter

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
n = int(f.readline())
a = []
for i in range(n):
    data = f.readline()
    a.append([int(data.split()[0]), int(data.split()[1])])

a.sort(key=itemgetter(1))

end = 0
count = 0
for i in range(n):
    if a[i][0] >= end:
        end = a[i][1]
        count += 1

m.write(str(count))
f.close()
m.close()

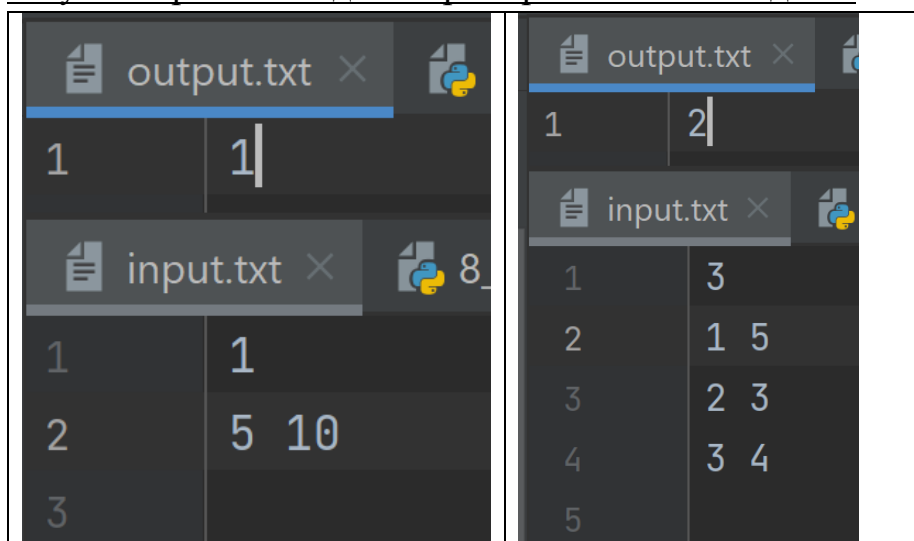
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

Сортируем лекции по возрастанию конца лекции. Далее проходимся по ним и при возможности добавляем их в наше расписание.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
--	----------------------------	----------------------

Минимальное значение	0.001535299999999968	14.140625
Пример №1	0.0011470999999999842	14.14453125
Пример №2	0.0027495999820530415	14.140625
Максимальное значение	0.0087291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №9. Распечатка (1 балл)

- Текст задачи:

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, афторефераты 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.

- Формат входного файла (input.txt).

На вход программе сначала подается число N – количество листов в заказе. В следующих 7 строках ввода находятся натуральные числа A1, A2, A3, A4, A5, A6, A7 соответственно.

- Ограничения на входные данные.

$$1 \leq N \leq 2 \times 10^9, 1 \leq A_i \leq 10^6$$

- Формат выходного файла (output.txt).

Выведите одно число – минимальную сумму денег в рублях, которая нужна для выполнения заказа. Гарантируется, что правильный ответ не будет превышать 2×10^9

- Ограничение по времени. 2сек.

- Ограничение по памяти. 256мб.

- Пример:

Input.txt	980	980
	1	1

	9	10
	90	100
	900	1000
	1000	900
	10000	10000
	10000	10000
Output.txt	882	900

Листинг кода:

```
import time
import os, psutil
from operator import itemgetter

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")

n = int(f.readline())
a = []
for i in range(7):
    data = f.readline()
    a.append(int(data))

for i in range(1, 7):
    a[i] = min(a[i], a[i-1]*10)
    print(a)

current = 0
result = 0
copy_n = n

while copy_n > 0:
    result += a[current]*(copy_n % 10)
    copy_n //= 10
    current += 1

power = 1
for i in range(7):
    if power >= n and a[i] < result:
        result = a[i]
    power *= 10

m.write(str(result))
f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

Сначала считаем минимум для печати количества листов за которые указана цена(это или цена указанная или сумма несколько предыдущих

цен). После этого считаем цену и восстанавливаем последовательность, в которой мы заменяли.

Результат работы кода на примерах из текста задачи:

output.txt ×
main.py

1
382

input.txt ×

1
980
2
1
3
9
4
90
5
900
6
1000
7
10000
8
10000
9

output.txt ×

1
900

input.txt ×

1
980
2
1
3
10
4
100
5
1000
6
900
7
10000
8
10000
9

Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.001535299999999968	14.140625
Пример №1	0.00041720003355294466	14.50390625
Пример №2	0.0004692000220529735	14.2265625
Максимальное значение	0.0087291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №10. Яблоки (1 балл)

- Текст задачи:

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

- Формат входного файла (input.txt).

В первой строке вводятся натуральные числа n и s – число яблок и начальный рост Алисы. В следующих n строках вводятся пары натуральных чисел a_i, b_i через пробел.

- Ограничения на входные данные.

$1 \leq n \leq 1000, 1 \leq s \leq 1000, 1 \leq a_i, b_i \leq 1000.$

- Формат выходного файла (output.txt).

Если яблоки съесть нельзя, выведите число -1. Иначе выведите n чисел – номера яблок, в том порядке, в котором их нужно есть.

- Ограничение по времени. 2сек.

- Ограничение по памяти. 256 мб.

- Пример:

Input.txt	3 5 2 3 10 5 5 10	3 5 2 3 10 5 5 6
Output.txt	1 3 2	-1

Листинг кода:

```
import time
import os, psutil

def eating_apples(apples, n, s):
    apples.sort(key=lambda a: [-(a[1] - a[0]), a[0]])
    j = 0
    psbl = True
    order = []
    while psbl and j < n:
        found = False
        a = 0
        while a < n and not found:
```

```

        if s - apples[a][0] > 0 and apples[a][2] != 0:
            found = True
            s += apples[a][1] - apples[a][0]
            order.append(apples[a][2])
            apples[a][2] = 0

        a += 1
        psbl = found
        j += 1
    if not psbl:
        order = []
    return order

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
n, s = map(int, f.readline().split())
apples = []
for i in range(n):
    a, b = map(int, f.readline().split())
    apples.append([a, b, i + 1])

answer = eating_apples(apples, n, s)
if len(answer) == 0:
    m.write("-1")
else:
    for i in range(len(answer)):
        m.write(str(answer[i]) + ' ')
f.close()
m.close()

print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

Мы читаем данные из файла, а далее сортируем их по 2 ступеням, первая разница (в приросте и снижение роста), а второй снижение роста. Далее мы проходимся по всем элементам и пытаемся съесть яблоки, если у нас не получится это сделать в данном порядке, то не получится вообще.

Результат работы кода на примерах из текста задачи:

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

Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.008535299999999968	14.140625
Пример №1	0.0011470999999999842	14.14453125
Пример №2	0.0019332999363541603	14.5859375
Максимальное значение	0.0017291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №12. Последовательность (1 балл)

- Текст задачи:

Дана последовательность натуральных чисел a_1, a_2, \dots, a_n , известно, что $a_i \leq i$ для любого $1 \leq i \leq n$. 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.

- Формат входного файла (input.txt).

В первой строке входного файла находится одно целое число n . Во второй строке находится n целых чисел a_1, a_2, \dots, a_n .

- Ограничения на входные данные.

$1 \leq n \leq 40000, 1 \leq a_i \leq i$.

- Формат выходного файла (output.txt).

В первую строку выходного файла выведите количество элементов последовательности в любой из получившихся двух частей, а во вторую строку через пробел номера этих элементов. Если построить такое разбиение невозможно, выведите -1.

- Ограничение по времени. 2сек.

- Ограничение по памяти. 256 мб.

- Пример:

Input.txt	3 1 2 3
Output.txt	1 3

Листинг кода:

```
import time
import os, psutil

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
n = int(f.readline())
numbers = list(map(int, f.readline().split()))

res = 0
if sum(numbers) % 2 == 0:
    half_sum = sum(numbers) / 2
    numbers.sort()
    numbers.reverse()
    part = []
    j = 0
```

```

while sum(part) != half_sum:
    if j > len(numbers) - 1:
        res = -1
        break
    if (sum(part) + numbers[j]) <= half_sum:
        part.append(numbers[j])
        del (numbers[j])
    else:
        j += 1
else:
    res = -1

if res == -1:
    m.write('-1')
else:
    m.write(str(len(part)) + '\n')
    m.write(' '.join(map(str, part)))

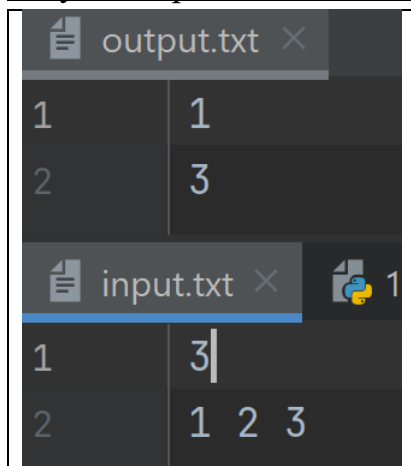
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

Мы читаем данные из файла, а далее сортируем их по убыванию. Далее пытаемся их добавить, добавляем, если сумма не превысит половины от суммы последовательности. Так проходимся по всем элементам. Если часть суммы равна половине последовательности, то это ответ, иначе невозможно разделить.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное	0.008535299999999968	14.140625

значение		
Пример №1	0.0007286999998541432	14.46484375
Максимальное значение	0.0017291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №13. Сувениры (1.5 балла)

- Текст задачи:

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

- Формат входного файла (input.txt).

В первой строке дано целое число n . Во второй строке даны целые числа v_1, v_2, \dots, v_n , разделенные пробелами.

- Ограничения на входные данные.

$1 \leq n \leq 20, 1 \leq v_i \leq 30$ для всех i .

- Формат выходного файла (output.txt).

Выведите 1, если можно разбить v_1, v_2, \dots, v_n на три подмножества с одинаковыми суммами и 0 в противном случае.

- Ограничение по времени. 5сек.

Пример:

Input.txt	4 3 3 3 3	1 40	11 17 59 34 57 17 23 67 1 18 2 59	13 1 2 3 4 5 5 7 7 8 10 12 19 25
Output.txt	0	0	1	1

Листинг кода:

```
import time
import os, psutil

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
n = int(f.readline())
numbers = list(map(int, f.readline().split()))

res = 1
if sum(numbers) % 3 == 0:
    third_part = sum(numbers) / 3
    numbers.sort()
    numbers.reverse()
    for i in range(3):
        part = []
        j = 0
        while sum(part) != third_part:
            if j > len(numbers)-1:
                res = 0
                break
            if (sum(part) + numbers[j]) <= third_part:
                part.append(numbers[j])
```

```

        del(numbers[j])
    else:
        j += 1
    if j > len(numbers) - 1:
        break
else:
    res = 0

if res == 1:
    res = 1
    if sum(numbers) % 2 == 0:
        half_sum = sum(numbers) / 2
        numbers.sort()
        numbers.reverse()
        part = []
        j = 0
        while sum(part) != half_sum:
            if j > len(numbers) - 1:
                res = 0
                break
            if (sum(part) + numbers[j]) <= half_sum:
                part.append(numbers[j])
                del (numbers[j])
            else:
                j += 1
        else:
            res = 0

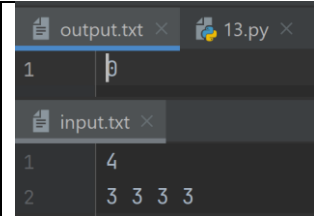
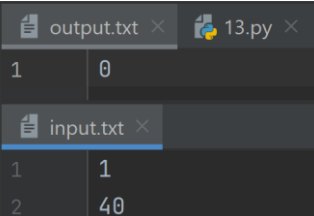
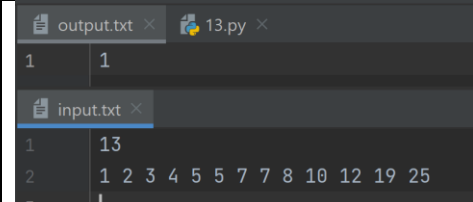
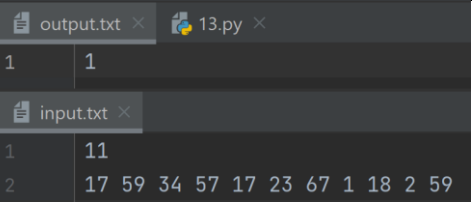
m.write(str(res))
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

Решение аналогично предыдущей задаче, только сначала рассматривается возможность получения трети, а затем из остатка половины.

Результат работы кода на примерах из текста задачи:

Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.008535299999999968	14.140625
Пример №1	0.0025200999999856029	14.484375
Пример №2	0.0019332999363541603	14.53515625
Пример №3	0.0003843000004053465	14.53515625
Пример №4	0.0018532000003688154	14.53515625
Максимальное значение	0.0017291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием жадных алгоритмов.

Задача №15. Удаление скобок (2 балла)

- Текст 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 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 выходного файла (output.txt).

Выведите строку максимальной длины, являющейся правильной скобочной последовательностью, которую можно получить из исходной строки удалением некоторых символов.

- Ограничение по времени. 2сек.

- Ограничение по памяти. 256 мб.

- Пример:

Input.txt	([)]
Output.txt	[]

Листинг кода:

```
import time
import os, psutil
import math

t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open("input.txt")
m = open("output.txt", "w")
s = f.readline()
n = len(s)
dp = []
pos = []
for i in range(n):
    dp.append([0] * n)
    pos.append([0] * n)

for i in range(n):
    for j in range(n):
        if i == j:
            dp[i][j] = 1
for right in range(n):
    for left in range(right, -1, -1):
```

```

        if left == right:
            dp[left][right] = 1
        else:
            _min = math.inf
            mink = -1
            b1 = s[left] == '(' and s[right] == ')'
            b2 = s[left] == '[' and s[right] == ']'
            b3 = s[left] == '{' and s[right] == '}'
            if b1 or b2 or b3:
                _min = dp[left + 1][right - 1]
            for k in range(left, right):
                if _min > dp[left][k] + dp[k + 1][right]:
                    _min = dp[left][k] + dp[k + 1][right]
                    mink = k
            dp[left][right] = _min
            pos[left][right] = mink

def rec(l, r):
    temp = r - l + 1
    if dp[l][r] == temp:
        return
    if pos[l][r] == -1:
        m.write(s[l])
        rec(l + 1, r - 1)
        m.write(s[r])
        return
    rec(l, pos[l][r])
    rec(pos[l][r] + 1, r)

rec(0, n - 1)

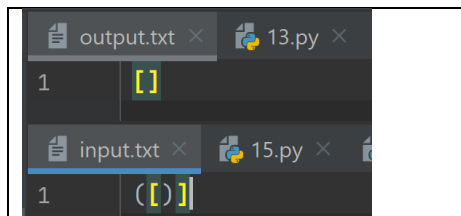
f.close()
m.close()
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")

```

Текстовое объяснение решения:

Наша программа состоит из двух функций первая считает и расставляет, какие скобки надо оставить, а втора восстанавливает последовательность и выводит результат. В первой создаётся две квадратные матрицы, размером длины последовательности. Мы проходимся по скобочной последовательности с двух сторон и сравниваем скобки. Если у нас не нужно удалять скобочки, то во втором массиве остается -1, по нему мы потом мы и восстанавливаем.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Минимальное значение	0.008535299999999968	14.140625
Пример №1	0.0021655999989889096	14.37109375
Максимальное значение	0.0017291999999999863	14.171875

Вывод по задаче: Мы научились решать задачи с использованием динамического программирования.

Вывод

Я узнал, что такое динамическое программирование и жадные алгоритмы, научился решать наиболее часто встречающиеся случаи задач.