

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 12

Выполнил:  
Кузнецов А.Г.  
К3140

Проверила:  
Артамонова В.Е.

Санкт-Петербург  
2023 г.

## Содержание отчета

<b>Содержание отчета</b>	<b>2</b>
<b>Задачи по варианту</b>	<b>3</b>
Задача №2 Заправки.	3
Задача №6 Максимальная зарплата.	4
Задача №8 Расписание лекций.	6
Задача №16 Продавец.	7
Задача №17 Ход конём.	8
<b>Дополнительные задачи</b>	<b>10</b>
Задача №1 Максимальная стоимость добычи.	10
Задача №3 Максимальный доход от рекламы.	11
Задача №4 Сбор подписей.	12
Задача №5 Максимальное количество призов.	14
Задача №7 Проблема сапожника.	15
Задача №9 Распечатка.	16
Задача №10 Яблоки.	19
Задача №11 Максимальное количество золота.	20
Задача №12 Последовательность.	21
Задача №13 Сувениры.	22
<b>Вывод</b>	<b>24</b>
Задача №N М.	25

## Задачи по варианту

### Задача №2 Заправки.

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

- Формат ввода / 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  $stop1, stop2, \dots, stopn$ .
- Ограничения на входные данные.  $1 \leq d \leq 105, 1 \leq m \leq 400, 1 \leq n \leq 300, 1 < stop1 < stop2 < \dots < stopn < d$
- 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  $stop1, stop2, \dots, stopn$  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.

```
with open('input.txt', 'r') as f:
    d = int(f.readline().strip()) # Расстояние между городами в км
    m = int(f.readline().strip()) # Сколько может проехать км автомобиль
    # на полном баке
    n = int(f.readline().strip()) # Количество заправок на пути
    stops = list(map(int, f.readline().strip().split())) # Заправки

last_stop = 0 # Последняя остановка
count = 0 # Количество дозаправок
remaining_distance = d # Сколько нам осталось проехать

# Проверяем нужно ли нам вообще запраправляться
# Если можем проехать без дозаправок, то выводим 0
if d <= m:
    with open('output.txt', 'w') as f:
        f.write('0')
    exit()

for i in range(n):
    distance_to_next_stop = stops[i] - last_stop # Расстояние до следующей
    # заправки

    # Проверяем сможем ли вы проехать до следующей заправки на полном баке
    # Если нет, то выводим -1
```

```

if distance_to_next_stop > m:
    with open('output.txt', 'w') as f:
        f.write('-1')
    exit()

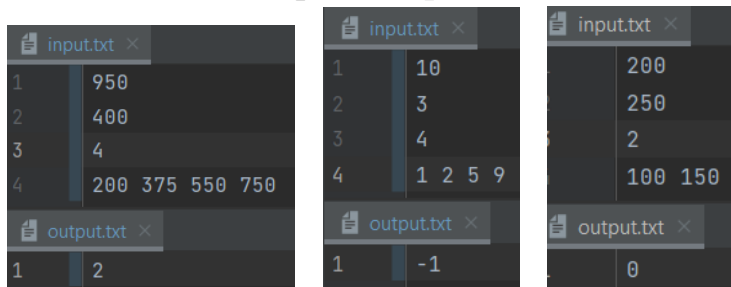
# Проверяем сможем ли мы проехать до следующей заправки без дозаправки
# Если нет, то заправляемся
elif distance_to_next_stop > remaining_distance - m:
    last_stop = stops[i]
    count += 1
    remaining_distance = d - last_stop

# Спокойно едем дальше
else:
    remaining_distance -= distance_to_next_stop

with open('output.txt', 'w') as f:
    f.write(str(count))

```

На вход из файла input.txt мы получаем следующие параметры: 1. Расстояние между городами в км, 2. Сколько км может проехать автомобиль на полном баке, 3. Количество заправок на пути, 4. Расстояния до каждой заправки. Далее мы проверяем частный случай, а именно мы проверяем нужно ли нам заправляться, если нет, то выводим 0, в противном случае идём дальше и проверяем уже с каждой заправкой нужно ли нам там заправляться или же нет и сможем ли мы доехать до заправки вовсе. Ответ мы записываем в файл output.txt



Вывод по 2 задаче: В ходе работы над второй задачей был написан алгоритм жадной сортировки, которая помогает определить нужно ли нам заправлять машину.

### Задача №6 Максимальная зарплата.

В качестве последнего вопроса успешного собеседования ваш начальник дает вам несколько листков бумаги с цифрами и просит составить из этих цифр наибольшее число. Полученное число будет вашей зарплатой, поэтому вы очень заинтересованы в максимизации этого числа. Как вы можете это сделать?

На лекциях мы рассмотрели следующий алгоритм составления наибольшего числа из заданных однозначных чисел.

К сожалению, этот алгоритм работает только в том случае, если вход состоит из однозначных чисел. Например, для ввода, состоящего из двух целых чисел 23 и 3 (23 не однозначное число!) возвращается 233, в то время как наибольшее число на самом деле равно 323. Другими словами, использование наибольшего числа из входных данных в качестве первого числа не является безопасным ходом.

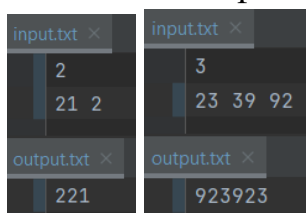
Ваша цель в этой задаче – настроить описанный выше алгоритм так, чтобы он работал не только с однозначными числами, но и с произвольными положительными целыми числами.

- Постановка задачи. Составить наибольшее число из набора целых чисел.
- Формат ввода / 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 вывода / выходного файла (output.txt). Выведите наибольшее число, которое можно составить из  $a_1, a_2, \dots, a_n$ .
- Ограничение по времени. 2 сек.

```
with open('input.txt', 'r') as f:
    n = int(f.readline().strip()) # Количество целых чисел
    a = list(map(str, f.readline().strip().split())) # Числа
    a.sort(key=lambda x: x * 5, reverse=True)
```

```
with open('output.txt', 'w') as f:
    f.write(''.join(map(str, a)))
```

На вход из файла input.txt мы получаем следующие параметры: 1. Количество чисел, 2. Сами числа. Далее мы сортируем числа, умножив их на 5 (Дело в том, что сортировка строк происходит посимвольно, что позволяет нам точно распределить числа), а потом записываем ответ мы записываем в файл output.txt.



Вывод по 6 задаче: В ходе работы над шестой задачей был реализован алгоритм, который из натуральных чисел составляет максимальное число

## Задача №8 Расписание лекций.

- Постановка 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 прочесть студентам. Чтобы студенты, в начале февраля, увидели расписание лекций, необходимо его составить прямо сейчас. И без вас нам здесь не справиться. У нас есть список заявок от преподавателей на лекции для одной из аудиторий. Каждая заявка представлена в виде временного интервала  $[s_i, f_i)$  - время начала и конца лекции. Лекция считается открытым интервалом, то есть какая-то лекция может начаться в момент окончания другой, без перерыва. Необходимо выбрать из этих заявок такое подмножество, чтобы суммарно выполнить максимальное количество заявок. Учтите, что одновременно в лекционной аудитории, конечно же, может читаться лишь одна лекция.
- Формат ввода / 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 вывода / выходного файла (output.txt). Выведите одно число – максимальное количество заявок на проведение лекций, которые можно выполнить.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

```
def takeSecond(elem):  
    return elem[1]  
  
with open('input.txt', 'r') as f:  
    n = int(f.readline().strip()) # Количество лекций  
    time = [] # Список начала и конца лекций  
    for i in range(n):  
        time.append(list(map(int, f.readline().strip().split())))  
    time.sort(key=takeSecond) # Сортировка по второму элементу  
  
last_finish = 0  
count = 0  
  
for start, finish in time:  
    # Если начало лекции не накладывается на конец другой, то  
    # добавляем лекцию к общему количеству и  
    # обновляем конечное время последней лекции
```

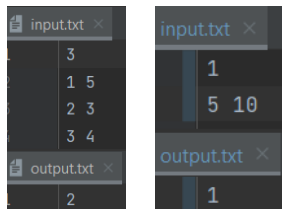
```

if start >= last_finish:
    count += 1
    last_finish = finish

with open('output.txt', 'w') as f:
    f.write(str(count))

```

На вход из файла input.txt мы получаем следующие параметры: 1. Количество лекций, 2. Список лекций с началом и концом. Сортируем список лекций по второму элементу, а потом проходимся по лекциям и проверяем если начало лекции не накладывается на конец другой, то добавляем лекцию к общему количеству и обновляем конечное время последней лекции. Ответ мы записываем в файл output.txt.



Вывод по 8 задаче: В ходе работы над восьмой задачей был реализован алгоритм, который покрывает отрезками пространство с минимальными пробелами между ними

## Задача №16 Продавец.

- Постановка задачи. Продавец техники хочет объехать  $n$  городов, посетив каждый из них ровно один раз. Помогите ему найти кратчайший путь.
- Формат ввода / 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.  $1 \leq n \leq 13$ ,  $0 \leq a_{i,j} \leq 106$ ,  $a_{i,j} = a_{j,i}$ ,  $a_{i,i} = 0$ .
- Формат вывода / 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. 1 сек.
- Ограничение по памяти. 256 мб.

```

import itertools

with open('input.txt', 'r') as f:
    n = int(f.readline().strip()) # Количество городов
    distances = [] # Длины между городами
    for _ in range(n):
        distances.append(list(map(int, f.readline().strip().split())))

```

```

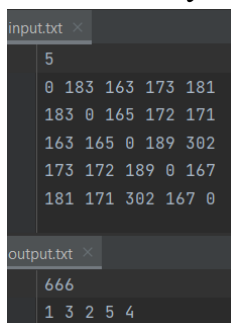
min_distance = 10 ** (6 + n + 1)
min_path = 0

# Генерация путей
for path in itertools.permutations(range(n)):
    # Считаем расстояние между городами
    distance = 0
    for i in range(n - 1):
        distance += distances[path[i]][path[i + 1]]
        if distance > min_distance:
            break
    # Обновление минимальной дистанции и минимального пути
    if distance < min_distance:
        min_distance = distance
        min_path = path

with open('output.txt', 'w') as f:
    f.write(str(min_distance) + '\n')
    f.write(' '.join(str(x + 1) for x in min_path))

```

На вход из файла input.txt мы получаем следующие параметры: 1. Количество городов, 2. Длины путей между городами. Далее генерируем возможные пути и считаем расстояние полного пути. Находим минимальную дистанцию и ответ записываем в файл output.txt.



```

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
1 3 2 5 4

```

Вывод по 16 задаче: В ходе работы над шестнадцатой был реализован алгоритм, который выводит минимальную дистанцию и города, которые нужно пройти

## Задача №17 Ход конём.

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

Напишите программу, определяющую количество телефонных номеров длины N, набираемых ходом коня. Поскольку таких номеров может быть очень много, выведите ответ по модулю 109 .

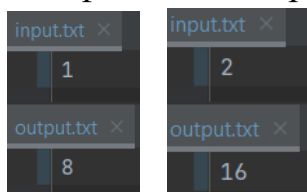


- Формат ввода / входного файла (input.txt). Во входном файле записано одно целое число N.
- Ограничения на входные данные.  $1 \leq N \leq 1000$ .
- Формат вывода / выходного файла (output.txt). Выведите в выходной файл искомое количество телефонных номеров по модулю 109.
- Ограничение по времени. 1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

```
with open("input.txt", "r") as f:
    n = int(f.readline().strip())
    result = 0
dp = [[0 for j in range(n + 1)] for i in range(10)]
for i in range(10):
    dp[i][1] = 1
for j in range(2, n + 1):
    for i in range(10):
        if i == 0:
            dp[0][j] = (dp[4][j - 1] + dp[6][j - 1])
        elif i == 1:
            dp[1][j] = (dp[6][j - 1] + dp[8][j - 1])
        elif i == 2:
            dp[2][j] = (dp[7][j - 1] + dp[9][j - 1])
        elif i == 3:
            dp[3][j] = (dp[4][j - 1] + dp[8][j - 1])
        elif i == 4:
            dp[4][j] = (dp[0][j - 1] + dp[3][j - 1] + dp[9][j - 1])
        elif i == 6:
            dp[6][j] = (dp[0][j - 1] + dp[1][j - 1] + dp[7][j - 1])
        elif i == 7:
            dp[7][j] = (dp[2][j - 1] + dp[6][j - 1])
        elif i == 8:
            dp[8][j] = (dp[1][j - 1] + dp[3][j - 1])
        elif i == 9:
            dp[9][j] = (dp[2][j - 1] + dp[4][j - 1])

for i in range(1, 10):
    if i != 8:
        result += dp[i][n]
with open("output.txt", "w") as f:
    f.write(str(result))
```

На вход из файла input.txt мы получаем длину телефонного номера. Далее создаём двумерных массив и заполняем его перебирая значения возможных телефонных номеров. Ответ записываем в файл output.txt.



Вывод по 17 задаче: В ходе работы над семнадцатой задачи был реализован код, который находит количество телефонных номеров, которые бы набирались на кнопочном телефоне ходом коня длиной n.

## Дополнительные задачи

### Задача №1 Максимальная стоимость добычи.

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.  $1 \leq n \leq 103$ ,  $0 \leq W \leq 2 \cdot 10^6$ ,  $0 \leq p_i \leq 2 \cdot 10^6$ ,  $0 \leq w_i \leq 2 \cdot 10^6$  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.

```
with open('input.txt', 'r') as f:
    n, W = map(int, f.readline().split()) # n - количество предметов, W -
    # вместимость сумки
    parameters = [] # цена за 1 единицу веса, вес, цена
    for _ in range(n):
        p, w = map(int, f.readline().split())
        parameters.append((p / w, w, p))
    parameters.sort(reverse=True)
    total_value = 0

for ratio, weight, price in parameters:
    # Если вместимость 0, то выводим ответ 0
    if W == 0:
        break
    # Если мы можем положить в рюкзак полностью предмет,
    # то мы так и делаем
    elif W >= weight:
        total_value += price
        W -= weight
    # Если мы не можем положить в рюкзак полностью предмет,
    # то делим его и помещаем в рюкзак дробную часть
    else:
```

```

fraction = W / weight
total_value += price * fraction
W = W - (weight * fraction)

```

```

with open('output.txt', 'w') as f:
    f.write(str(round(total_value, 4)))

```

На вход из файла input.txt мы получаем следующие значения: 1. Количество предметов, вместимость сумки, 2. Значение веса и стоимость предмета. Также создаём переменную соотношения цены к весу. Сортируем предметы по соотношению цены к весу и помещаем в рюкзак целые или дробные предметы. Ответ записываем в файл output.txt.

input.txt	3 50 60 20 100 50 120 30
output.txt	180

input.txt	1 10 500 30
output.txt	166.6667

Вывод по 1 задаче: В ходе работы над первой задачей был реализован алгоритм, который решает задачу о дробном рюкзаке

### Задача №3 Максимальный доход от рекламы.

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.

- Постановка задачи. 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 ввода / 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.  $1 \leq n \leq 103$ ,  $-105 \leq a_i, b_i \leq 105$ , для всех  $1 \leq i \leq n$ .
- Формат вывода / 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. 2 сек.

```

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

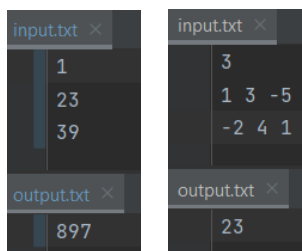
```

```

n = int(f.readline().strip()) # Количество объявлений
a = list(map(int, f.readline().strip().split())) # Прибыль за клик по
i-му объявлению
b = list(map(int, f.readline().strip().split())) # Среднее количество
кликов в день i-го слота
a.sort(reverse=True)
b.sort(reverse=True)
result = 0
for i in range(n):
    result += a[i] * b[i]
with open('output.txt', 'w') as f:
    f.write(str(result))

```

На вход из файла input.txt мы получаем следующие значения: 1. Количество объявлений, 2. Прибыль за клик по i-му объявлению, 3. Среднее количество кликов в день i-го слота. Сортируем массивы с прибылью и с средним количеством кликов в день по убыванию. Перемножаем значения начиная с минимальных. Ответ записываем в файл output.txt.



Вывод по 3 задаче: В ходе работы над третьим заданием был реализован алгоритм по распределению рекламы по слотам, чтобы максимизировать общий доход

#### Задача №4 Сбор подписей.

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.

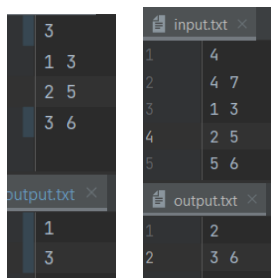
- Постановка задачи. Дан набор из  $n$  отрезков  $[a_0, b_0], [a_1, b_1], \dots, [a_{n-1}, b_{n-1}]$  с координатами на прямой, найдите минимальное количество  $m$  точек такое, чтобы каждый отрезок содержал хотя бы одну точку. То есть найдите набор целых чисел  $X$  минимального размера такой, чтобы 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.  $1 \leq n \leq 102$ ,  $0 \leq a_i, b_i \leq 109$  - 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

```
def takeSecond(elem):  
    return elem[1]  
  
with open('input.txt', 'r') as f:  
    n = int(f.readline().strip()) # Количество отрезков n  
    segments = [] # Список для хранения отрезков  
    for _ in range(n):  
        a, b = map(int, f.readline().split())  
        segments.append((a, b))  
  
segments.sort(key=takeSecond) # Сортировка по второму элементу  
  
points = [] # список точек, которые будем использовать для покрытия  
отрезков  
current_point = segments[0][1] # начинаем с конца первого отрезка  
points.append(current_point)  
  
# проходимся по оставшимся отрезкам  
for i in range(1, n):  
    # Если начало следующего отрезка больше текущей точки,  
    # то добавляем его конец  
    if segments[i][0] > current_point:  
        current_point = segments[i][1]  
        points.append(current_point)  
  
with open('output.txt', 'w') as f:  
    f.write(str(len(points)) + '\n')  
    for point in points:  
        f.write(str(point) + ' ')
```

На вход из файла input.txt мы получаем следующие значения: 1. Количество отрезков, 2. Сами отрезки. Сортируем отрезки по второму значению. Начинаем с конца первого отрезка и проходимся по оставшимся отрезкам. Если начало следующего отрезка больше текущей точки, то добавляем его конец. Ответ записываем в файл output.txt.



Вывод по 4 задаче: В ходе работы над второй задачей был реализован алгоритм, который помогает отметить как можно меньше точек на прямой так, чтобы каждый отрезок содержал хотя бы одну отмеченную точку.

### Задача №5 Максимальное количество призов.

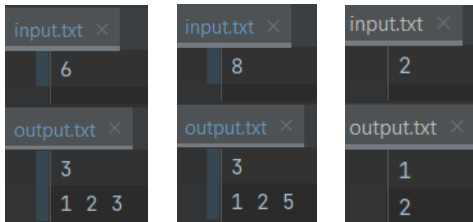
Вы организуете веселый конкурс для детей. В качестве призового фонда у вас есть  $n$  конфет. Вы хотели бы использовать эти конфеты для раздачи  $k$  лучшим местам в конкурсе с естественным ограничением, заключающимся в том, что чем выше место, тем больше конфет. Чтобы осчастливить как можно больше детей,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

- Постановка 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 ввода / входного файла (input.txt). Входные данные состоят из одного целого числа  $n$ .
- Ограничения на входные данные.  $1 \leq n \leq 10^9$ .
- Формат вывода / 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. 2 сек.

```
with open('input.txt', 'r') as f:
    n = int(f.readline().strip()) # Количество конфет
k, s = 1, 0
numbers = [] # Призы
while n >= s + k:
    numbers.append(k)
    s += k
    k += 1
if n - s > 0:
    numbers[-1] += n - s
with open('output.txt', 'w') as f:
```

```
f.write(str(len(numbers)) + '\n')
for number in numbers:
    f.write(str(number) + ' ')
```

На вход из файла input.txt мы получаем количество конфет. Создаём два переменные k и s. Так как нам нужно наибольшее количество различных натуральных чисел, то начнём с единицы и будем добавлять эти числа в массив numbers. Далее проверяем есть у нас ещё конфеты, если да, то добавляем остаток к последнему значению. Ответ записываем в файл output.txt.



Вывод по 5 задаче: В ходе работы над пятой задачей был реализован алгоритм, который находит наибольшее количество различных натуральных чисел, которые в сумме дают n.

## Задача №7 Проблема сапожника.

- Постановка задачи. В некоей воинской части есть сапожник. Рабочий день сапожника длится K минут. Заведующий складом оценивает работу сапожника по количеству починенной обуви, независимо от того, насколько сложный ремонт требовался в каждом случае. Дано n сапог, нуждающихся в починке. Определите, какое максимальное количество из них сапожник сможет починить за один рабочий день.
- Формат ввода / 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 вывода / выходного файла (output.txt). Выведите одно число – максимальное количество сапог, которые можно починить за один рабочий день.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

```
with open('input.txt', 'r') as f:
    K, n = map(int, f.readline().split()) # Рабочий день K минут, n сапог
    t = list(map(int, f.readline().strip().split())) # Количество минут,
    # чтобы починить i-й сапог.
```

```

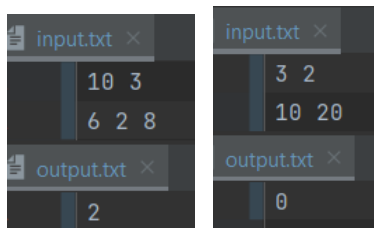
t.sort()
boots = 0 # Количество сапог

for i in range(n):
    if K >= t[i]:
        K -= t[i]
        boots += 1
    else:
        break

with open('output.txt', 'w') as f:
    f.write(str(boots))

```

На вход из файла input.txt мы получаем следующие значения: 1. Время рабочего дня, 2. Количество сапог, 3. Количество минут, которые требуются, чтобы починить i-й сапог. Сортируем количество минут по возрастанию и проверяем от минимального к максимального сколько можно починить сапог. Ответ записываем в файл output.txt.



Вывод по 7 задаче: В ходе работы над седьмой задачей был написан код, в котором мы находим максимальное количество сапог, которые может починить сапожник за рабочий день.

## Задача №9 Распечатка.

- Постановка 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, авторефераты для защиты и т.п.). Вы оценили объем 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 ввода / входного файла (input.txt). На вход программе сначала подается число  $N$  – количество листов в заказе. В следующих 7 строках ввода находятся натуральные числа  $A_1, A_2, A_3, A_4, A_5, A_6, A_7$  соответственно.
- Ограничения на входные данные.  $1 \leq N \leq 2 \times 10^9, 1 \leq A_i \leq 10^6$
- Формат вывода / выходного файла (output.txt). Выведите одно число – минимальную сумму денег в рублях, которая нужна для выполнения заказа. Гарантируется, что правильный ответ не будет превышать  $2 \times 10^9$ .
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

*# Находим минимальную цену, если заказываем более крупную партию*

```
def Big_Batch(N, A1, A2, A3, A4, A5, A6, A7):
    cost = 2 * 10 ** 10
    if 100000 < N <= 1000000:
        cost = A7
    elif 10000 < N <= 100000:
        cost = min(A6, A7)
    elif 1000 < N <= 10000:
        cost = min(A5, A6, A7)
    elif 100 < N <= 1000:
        cost = min(A4, A5, A6, A7)
    elif 10 < N <= 100:
        cost = min(A3, A4, A5, A6, A7)
    elif 1 < N <= 10:
        cost = min(A2, A3, A4, A5, A6, A7)
    return cost
```

*# Находим минимальную цену, если заказываем с помощью комбинирования тарифов*

```
def Combination(N, A1, A2, A3, A4, A5, A6, A7):
    cost = 0
    if N <= 9:
        cost = N * A1
    elif N <= 99:
        cost = ((N // 10) * A2) + ((N % 10) * A1)
    elif N <= 999:
        cost = ((N // 100) * A3) + (((N % 100) // 10) * A2) + ((N % 10) *
A1)
    elif N <= 9999:
        cost = ((N // 1000) * A4) + (((N % 1000) // 100) * A3) + (((N %
100) // 10) * A2) + ((N % 10) * A1)
    elif N <= 99999:
        cost = ((N // 10000) * A5) + (((N % 10000) // 1000) * A4) + (((N %
1000) // 100) * A3) + (((N % 100) // 10) * A2) + ((N % 10) * A1)
```

```

        cost = ((N // 10000) * A5) + (((N % 10000) // 1000) * A4) + (((N % 1000) // 100) * A3) + (
            ((N % 100) // 10) * A2) + ((N % 10) * A1)
    elif N <= 999999:
        cost = ((N // 100000) * A6) + (((N % 100000) // 10000) * A5) + (((N % 10000) // 1000) * A4) + (
            ((N % 1000) // 100) * A3) + (((N % 100) // 10) * A2) + ((N % 10) * A1)
    else:
        cost = ((N // 1000000) * A7) + (((N % 1000000) // 100000) * A6) + (((N % 100000) // 10000) * A5) + (
            ((N % 10000) // 1000) * A4) + (((N % 1000) // 100) * A3) + (((N % 100) // 10) * A2) + (
                (N % 10) * A1)

    return cost

```

```

with open('input.txt', 'r') as f:
    N = int(f.readline().strip())
    A1 = int(f.readline().strip())
    A2 = int(f.readline().strip())
    A3 = int(f.readline().strip())
    A4 = int(f.readline().strip())
    A5 = int(f.readline().strip())
    A6 = int(f.readline().strip())
    A7 = int(f.readline().strip())

```

```

with open('output.txt', 'w') as f:
    f.write(str(min(Big_Batch(N, A1, A2, A3, A4, A5, A6, A7),
        Combination(N, A1, A2, A3, A4, A5, A6, A7))))

```

На вход из файла input.txt мы получаем количество листов. У нас есть две функции: первая находит минимальное значение при комбинировании тарифов, вторая находит минимальное значение при покупке крупной партии. В конце сравниваем выводимые значение и ответ записываем в файл output.txt.

input.txt	output.txt
1 980	1 882
2 1	2 900
3 9	
4 90	
5 900	
6 1000	
7 10000	
8 10000	

Вывод по 9 задаче: В ходе работы над девятой задачей был реализован алгоритм, который находит минимальную сумму денег, которую мы можем потратить, чтобы распечатать N листов

## Задача №10 Яблоки.

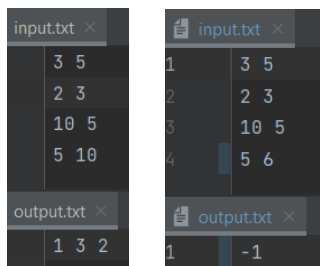
- Постановка задачи. Алисе в стране чудес попались  $n$  волшебных яблок. Про каждое яблоко известно, что после того, как его съешь, твой рост сначала уменьшится на  $a_i$  сантиметров, а потом увеличится на  $b_i$  сантиметров. Алиса очень голодная и хочет съесть все  $n$  яблок, но боится, что в какой-то 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 ввода / 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 вывода / 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. 2 сек.
- Ограничение по памяти. 256 мб.

```
with open('input.txt', 'r') as f:
    n, s = map(int, f.readline().split()) # Количество яблок и рост Алисы
    apples = [] # Список с яблоками и их свойствами
    for i in range(n):
        a, b = map(int, f.readline().split())
        apples.append((a, b, i + 1))
    apples.sort(key=lambda x: (x[0], -x[1])) # Сортируем яблоки по эффекту
    # уменьшения и увеличения

order = []
for a, b, i in apples:
    # Проверяем сможет ли Алиса съесть яблоко при этом,
    # чтобы её рост не стал меньше или равен нулю
    if s <= a:
        with open('output.txt', 'w') as f:
            f.write('-1')
        exit()
    s += (b - a) # Добавляем разницу эффекта уменьшения и увеличения
    order.append(str(i))
with open('output.txt', 'w') as f:
    f.write(' '.join(order))
```

На вход из файла input.txt мы получаем следующие значения: 1. Количество яблок, 2. Рост Алисы, 3. Свойства яблок. Сортируем список яблок по эффекту уменьшения и увеличения. Далее проверяем сможет ли Алиса съесть яблоко при этом, чтобы её рост не стал меньше или равен нулю. Если

такое возможно, то к росту добавляем разницу эффектов и записываем в последовательность  $i$ -е яблоко. Ответ записываем в файл output.txt.



Вывод по 10 задаче: В ходе работы над десятой задачей был реализован алгоритм нахождения последовательности, в которой Алисе следует съесть яблоки, чтобы ни разу её рост не стал меньше или равен нулю.

### Задача №11 Максимальное количество золота.

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  $w_0, w_1, \dots, w_{n-1}$ , определяющие вес золотых слитков.
- Ограничения на входные данные.  $1 \leq W \leq 104$ ,  $1 \leq n \leq 300$ ,  $0 \leq w_0, \dots, w_{n-1} \leq 105$
- Формат вывода / выходного файла (output.txt). Выведите максимальный вес золота, который поместится в сумку вместимости  $W$ .
- Ограничение по времени. 5 сек.

```
with open('input.txt', 'r') as f:
    W, n = map(int, f.readline().split()) # Вместимость сумки и количество
    # ЗОЛОТЫХ СЛИТКОВ
    w = list(map(int, f.readline().strip().split())) # Вес золотых слитков

dp = [[0] * (W+1) for _ in range(n+1)]

for i in range(1, n+1):
    for j in range(1, W+1):
        dp[i][j] = dp[i-1][j] # Если i-й слиток не положили
        if j >= w[i-1]: # Если i-й слиток положили
            dp[i][j] = max(dp[i][j], dp[i-1][j-w[i-1]] + w[i-1])

with open("output.txt", 'w') as f:
    f.write(str(dp[n][W]))
```

На вход из файла input.txt мы получаем следующие значения: 1. Вместимость сумки, 2. Количество золотых слитков, 3. Вес золотых слитков. Создаём двумерный массив размером  $(n+1) \times (W+1)$ , где  $dp[i][j]$  - максимальный вес золота, который может быть положен в рюкзак вместимости  $j$  с использованием первых  $i$  золотых слитков. Если мы не берем  $i$ -ый слиток, то максимальный вес не меняется (остается таким же, как для  $i-1$  слитков). А если мы берем  $i$ -ый слиток, то максимальный вес будет равен сумме веса  $i$ -го слитка и максимального веса, который может быть положен в рюкзак с использованием  $i-1$  слитков и вместимостью  $j-w[i-1]$ . Ответ записываем в файл output.txt.

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

Вывод по 11 задаче: В ходе работы над одиннадцатой задачи был реализован код, который решает задачу о рюкзаке методом динамического программирования.

### Задача №12 Последовательность.

- Постановка задачи. Дана последовательность натуральных чисел  $a_1, a_2, \dots, a_n$ , и 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 ввода / 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 вывода / 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. 2 сек.
- Ограничение по памяти. 256 мб.

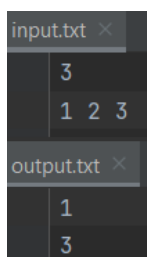
```
with open("input.txt", "r") as f:
    n = int(f.readline().strip())
    A = list(map(int, f.readline().strip().split()))
    A.sort(reverse=True)
```

```

s = sum(A)
p, q = [], []
if s % 2 != 0 or n == 1:
    with open("output.txt", "w") as f:
        f.write("-1")
    exit()
else:
    for a in A:
        if sum(p) + a <= s // 2:
            p.append(a)
        else:
            q.append(a)
    if sum(p) == sum(q):
        if len(p) <= len(q):
            with open("output.txt", "w") as f:
                f.write(str(len(p)) + '\n')
                for number in p:
                    f.write(str(number) + ' ')
            else:
                with open("output.txt", "w") as f:
                    f.write(str(len(q)) + '\n')
                    for number in q:
                        f.write(str(number) + ' ')
        else:
            with open("output.txt", "w") as f:
                f.write("-1")

```

На вход из файла input.txt мы получаем следующие значения: 1. Целое число n, 2. n целых чисел. Код находит сумму n целых чисел, смотрит четная ли сумма и если сумма чётная, то мы начинаем разбивать числа на два массива так, чтобы в двух массивах была одинаковая сумма. Ответ записываем в файл output.txt.



Вывод по 12 задаче: В ходе работы над двенадцатой задачей был реализован код, который разделяет целые числа на две части таким образом, что сумма элементов в каждой из частей будет равна половине суммы всех элементов последовательности.

### Задача №13 Сувениры.

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  $v_1, v_2, \dots, v_n$ , 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  $v_1, v_2, \dots, v_n$  на три подмножества с одинаковыми суммами и 0 в противном случае.
- Ограничение по времени. 5 сек.

```
with open("input.txt", "r") as f:
    n = int(f.readline().strip())
    A = list(map(int, f.readline().strip().split()))
    A.sort(reverse=True)
    s = sum(A)
    x, y, z = [], [], []
if s % 3 != 0 or n <= 2:
    with open("output.txt", "w") as f:
        f.write("0")
    exit()
else:
    for a in A:
        if sum(x) + a <= s // 3:
            x.append(a)
        elif sum(y) + a <= s // 3:
            y.append(a)
        else:
            z.append(a)
if sum(x) == sum(y) == sum(z):
    with open("output.txt", "w") as f:
        f.write("1")
else:
    with open("output.txt", "w") as f:
        f.write("0")
```

На вход из файла input.txt мы получаем следующие значения: 1. Целое число  $n$ , 2.  $n$  целых чисел. Код находит сумму  $n$  целых чисел, смотрит делится ли сумма на 3, если делится, то мы начинаем разбивать числа на три массива так, чтобы в трёх массивах была одинаковая сумма. Ответ записываем в файл output.txt.

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

Вывод по 13 задаче: В ходе работы над тринадцатой задачей был реализован код, который разделяет целые числа на три части таким образом, что сумма элементов в каждой из частей будет равна трети суммы всех элементов последовательности.

## **Вывод**

В ходе лабораторной работы была проведена работа с жадными алгоритмами и динамическим программированием.