Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(Университет ИТМО)

Факультет **Инфокоммуникационных технологий**

Образовательная программа **Интеллектуальные системы в гуманитарной сфере**

Направление подготовки **45.03.04 Интеллектуальные системы в гуманитарной сфере**

О Т Ч Е Т

лабораторной работе 1

на тему: “Жадные алгоритмы. Динамическое программирование No2”

Обучающийся Королева Екатерина
К3143

Работа выполнена с оценкой ____

Преподаватель:

_____
(подпись)

Дата 21.06.2022

Санкт-Петербург, 2021

1 задача. Максимальная стоимость добычи (0.5 балла)

Вор находит гораздо больше добычи, чем может поместиться в его сумке. Помогите ему найти самую ценную комбинацию предметов, предполагая, что любая часть предмета добычи может быть помещена в его сумку.

Цель - реализовать алгоритм для задачи о дробном рюкзаке.

- **Формат ввода / входного файла (input.txt).** В первой строке входных данных задано целое число n - количество предметов, и W - вместимость сумки. Следующие n строк определяют значения веса и стоимости предметов. В i-ой строке содержатся целые числа 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.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def InsertionSort (arr, check, w, p):
    for i in range(1, check):
```

```
        j = i-1
        while j >=0 and arr[j+1] > arr[j] :
            arr[j], arr[j+1] = arr[j+1], arr[j]
            w[j], w[j+1] = w[j+1], w[j]
            p[j], p[j+1] = p[j+1], p[j]
            j -= 1
    return arr, w, p

def Knapsack(W, w, p):
    A = [0]*l
    V = 0
    for i in range(l):
        if W == 0:
            return V, A
        a = min(w[i], W)
        V = V + a*(p[i]/w[i])
        w[i] = w[i] - a
        A[i] = A[i] + a
        W = W - a
    return V, A

def toFixed(num, digits=0):
    return f"{num:.{digits}f}"


if __name__=='__main__':
  with open('input1.txt') as file:
      first = list(map(int, file.readline().split()))
      l = first[0]
      v = first[1]
      things = []
      weights, price = [], []
      for i in range(1, l+1):
          line = list(map(int,
file.readline().split()))
          price.append(line[0])
          weights.append(line[1])
          things.append(price[i-1]/weights[i-1])
```

```
    InsertionSort(things, l, weights, price)
    v, a = Knapsack(v, weights, price)
    otvet = toFixed(v, digits=4)

    with open('output.txt', 'wt') as f:
        f.write(otvet)

print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())

'''
Комментарий:
Мы используем жадный подход. Основная идея жадного
подхода состоит в том, чтобы рассчитать соотношение
стоимость / вес для каждого элемента
и отсортировать его на основе этого отношения. Затем
необходимо элемент с наибольшим соотношением и
добавлять их до тех пор, пока мы не сможем
добавить следующий элемент целиком, а в конце
добавить следующий элемент в максимально возможном
количестве. Это оптимальное решение задачи.

Стратегия такая:
1. Отсортировать список предметов по их стоимости на
единицу веса.
2. Добавлять в "рюкзак" самые ценные предметы, пока
не достигнется максимальный вес.
'''
```

```
Время работы (в секундах): 0.0007039000000000004
Память 4229, и пик 19464
```

6 задача. Максимальная зарплата (0.5 балла)

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.

```
def LargestNumber(Digits):
    answer = ''
    while Digits:
        maxDigit = float('-inf')
        for digit in Digits:
            if digit >= maxDigit:
                maxDigit = digit
        answer += str(maxDigit)
        Digits.remove(maxDigit)
    return answer
```

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 $a_1, a_2, ..., a_n$.
- **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 $a_1, a_2, ..., a_n$.
- Ограничение по времени. 2 сек.
- Пример:

input.txt	output.txt
2 21 2	221

input.txt	output.txt
3 23 39 92	923923

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()

def Largestnumber(a):
    a.sort(reverse=True)
    num = [str(x) for x in range(9, 1, -1)]
    otvet = ''
    for k in num:
        for i in a:
            if k == str(i)[0]:
                otvet += str(i)
    return otvet


f=open('input1.txt','r')
a=open('output.txt', 'w')
vivod = int(f.readline())
numbers = list(map(int, f.readline().split()))


if 1 <= vivod <= 10**2 and min(list(map(int,
numbers))) >= 1 and max(list(map(int, numbers))) <
10**3:
    a.write(str(Largestnumber(numbers)))
else:
    a.write('ошибка')

f.close()
a.close()
print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())
'''
ОШИБКА С ВЫВОДОМ ЧИСЕЛ
'''
```

13 задача. Сувениры (1.5 балла)

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

- **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.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def func(x: list):
    if sum(x) % 3 == 0:
        tmp = sum(x) / 3
        x.sort()
        x.reverse()
```

```
        elems = list()
        for i in range(3):
            subelements = []
            j = 0
            while (sum(subelements) != tmp):
                if j >= len(x):
                    return 0
                if ((sum(subelements) + x[j]) <= tmp):
                    subelements.append(x[j])
                    del (x[j])
                else:
                    j += 1
            print(subelements)
            elems.append(subelements)
        return int(sum(elems[0]) == sum(elems[1]) ==
sum(elems[2]))
    else:
        return 0


if __name__ == '__main__':
    print(func([17, 59, 34, 57, 17, 23, 67, 1, 18, 2,
59]))


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())


'''
Создадим функцию, в которой будем проверять, можно ли
разделить сувениры на троих. Сначала проверим,
делится ли сумма
на три в принципе. Если нет, возвращаем 0. Если да,
то в цикле for разбиваем множество сувениров на три
подмножества,
```

```
равных по сумме.
'''
Время работы (в секундах): 9.819999999999968e-05
Память 282, и пик 718
```

15 задача. Удаление скобок (2 балла)

- **Постановка задачи.** Дана строка, составленная из круглых, квадратных и фигурных скобок. Определите, какое наименьшее количество символов необходимо удалить из этой строки, чтобы оставшиеся символы образовывали правильную скобочную последовательность.
- **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
([)]	[]

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def result(amount, number, left, right, line):
    if amount[left][right] == right - left + 1:
        return 0
    elif amount[left][right] == 0:
        file2.write(line[left:right + 1])
        return 0
    elif number[left][right] == -1:
        file2.write(line[left])
        result(amount, number, left + 1, right - 1,
line)
        file2.write(line[right])
```

```
        return 0
    else:
        result(amount, number, left,
number[left][right], line)
        result(amount, number, number[left][right] +
1, right, line)


def bracket_sequence(line):
    d = {'(': ')', '{': '}', '[': ']'}
    n = len(line)
    amount = []
    number = []
    for _ in range(n):
        amount.append([0] * n)
        number.append([0] * n)
    for right in range(n):
        for left in range(right, -1, -1):
            if left == right:
                amount[left][right] = 1
            else:
                mn = float('inf')
                mk = -1
                if line[left] in ['(', '[', '{'] and
line[right] == d[line[left]]:
                    mn = amount[left + 1][right - 1]
                for k in range(left, right):
                    if amount[left][k] + amount[k +
1][right] < mn:
                        mn = amount[left][k] +
amount[k + 1][right]
                        mk = k
                amount[left][right] = mn
                number[left][right] = mk
    result(amount, number, 0, n - 1, line)


with open('input1.txt', 'r') as file1:
```

```
    s = file1.readline()

with open('output.txt', 'w') as file2:
    bracket_sequence(s)


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())


'''
ОШИБКА В ВЫВОДЕ - NONE
'''
```

18 задача. Кафе (2.5 балла)

- **Постановка задачи.** Около университета недавно открылось новое кафе, в котором действует следующая система скидок: при каждой покупке более чем на 100 рублей покупатель получает купон, дающий право на один бесплатный обед (при покупке на сумму 100 рублей и меньше такой купон покупатель не получает). Однажды вам на глаза попался прейскурант на ближайшие n дней. Внимательно его изучив, вы решили, что будете обедать в этом кафе все n дней, причем каждый день вы будете покупать в кафе ровно один обед. Однако стипендия у вас небольшая, и поэтому вы хотите по максимуму использовать предоставляемую систему скидок так, чтобы ваши суммарные затраты были минимальны. Требуется найти минимально возможную суммарную стоимость обедов и номера дней, в которые вам следует воспользоваться купонами.

- **Формат ввода / входного файла (input.txt).** В первой строке входного файла дается целое число n - количество дней. В каждой из последующих n строк дано одно неотрицательное целое число s_i – стоимость обеда в рублях на соответствующий день i.

- **Ограничения на входные данные.** $0 \leq n \leq 100$, $0 \leq s_i \leq 300$ для всех $0 \leq i \leq n$.

- **Формат вывода / выходного файла (output.txt).** В первой строке выдайте минимальную возможную суммарную стоимость обедов. Во второй строке выдайте два числа k_1 и k_2 – количество купонов, которые останутся у вас неиспользованными после этих n дней и количество использованных вами купонов соответственно. В последующих k_2 строках выдайте в возрастающем порядке номера дней, когда вам следует воспользоваться купонами. Если существует несколько решений с минимальной суммарной стоимостью, то выдайте то из них, в котором значение k_1 максимально (на случай, если вы когда-нибудь ещё решите заглянуть в это кафе). Если таких решений несколько, выведите любое из них.

- Ограничение по времени. 2 сек.

- Ограничение по памяти. 64 мб.

- Пример:

input.txt	output.txt
5 110 40 120 110 60	260 0 2 3 5

input.txt	output.txt
3 110 110 110	220 1 1 2

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def coupons(prices):
    n = len(prices)
    arr = []
    for k in range(n + 1):
        arr.append([float('inf')] * (n + 1))
    arr[0][0] = 0
    prices.insert(0, 0)
    for i in range(1, n + 1):
        for j in range(0, n):
            if prices[i] > 100:
                arr[i][j] = min(arr[i - 1][j - 1] +
prices[i], arr[i - 1][j + 1])
            else:
                arr[i][j] = min(arr[i - 1][j] +
prices[i], arr[i - 1][j + 1])
    mn = min(arr[n])
    k1 = arr[n].index(mn)
    j = k1
    i = n
    k2 = 0
    days = []
    while i > 0 or j > 0:
        if prices[i] > 100:
            if arr[i - 1][j - 1] + prices[i] <= arr[i
- 1][j + 1]:
                i -= 1
                j -= 1
            else:
                days.append(i)
                k2 += 1
                i -= 1
                j += 1
```

```
        else:
            if arr[i - 1][j] + prices[i] <= arr[i -
1][j + 1]:
                i -= 1
            else:
                days.append(i)
                k2 += 1
                i -= 1
                j += 1
    return mn, k1, k2, days


with open('input1.txt', 'r') as file1:
    n = int(file1.readline())
    prices = []
    for _ in range(n):
        prices.append(int(file1.readline()))

price, unused, used, days = coupons(prices)

with open('output.txt', 'w') as file2:
    file2.write(str(price) + '\n')
    file2.write(str(unused) + ' ' + str(used) + '\n')
    file2.write('\n'.join(map(str, days)))


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())

'''
Создадим функцию, в которой будем проверять стоимость
обеда и, если обед стоит больше 100 рублей, добавлять
купон. Далее
будем смотреть, сколько обедов и на какую сумму мы
можем купить, и сколько купонов потратить.
'''
```

```
Время работы (в секундах): 0.0008409999999999945
Память 3556, и пик 18000
```

Дополнительные задачи:

2 задача. Заправки (0.5 балла)

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

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


with open('input1.txt', 'r') as file:
   destination = int(file.readline())
   distance = int(file.readline())
   number_of_stations = int(file.readline())
   gas_statinon = [int(i) for i in
file.readline().split()]
   gas_statinon.insert(0, 0)
   gas_statinon.append(destination)


def min_gas(x, n, l):
    gas_refill = 0
    gas_current = 0
    while gas_current <= n:
          gas_last = gas_current
          while gas_current <= n and (x[gas_current
+ 1] - x[gas_last]) <= l:
               gas_current += 1
          if gas_current == gas_last:
               return -1
          if gas_current <= n:
               gas_refill += 1
    return gas_refill


with open('output.txt', 'w') as file:
   file.write(str(min_gas(gas_statinon,
number_of_stations, distance)))


print(min_gas(gas_statinon, number_of_stations,
distance))
```

```
print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())


'''
Создадим функцию, в которой будем проходить до города
B, текущая заправка будет последней заправкой,
и мы будем обновлять ее значение. Внутренний цикл
while будет проверкой - достаточно ли этой заправки,
чтобы доехать.
Будем делать так до тех пор, пока расстояние между
заправками можно преодолеть «безопасно» с нашим
бензобаком.
Если значение заправки не обновляется - добраться до
пункта назначения невозможно.
'''
```

```
Время работы (в секундах): 0.0006425000000000042
Память 1456, и пик 18009
```

3 задача. Максимальный доход от рекламы (0.5 балла)

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

- **Постановка задачи.** Даны две последовательности $a_1, a_2, ..., a_n$ (a_i - прибыль за клик по i-му объявлению) и $b_1, b_2, ..., b_n$ (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 $a_1, a_2, ..., a_n$, в третьей - последовательность целых чисел $b_1, b_2, ..., b_n$.
- **Ограничения на входные данные.** $1 \le n \le 10^3$, $-10^5 \le a_i, b_i \le 10^5$, для всех $1 \le i \le n$.
- **Формат вывода / выходного файла (output.txt).** Выведите максимальное значение $\sum_{i=1}^{n} a_i c_i$, где $c_1, c_2, ..., c_n$ является перестановкой $b_1, b_2, ..., b_n$.
- Ограничение по времени. 2 сек.
- Примеры:

input.txt	output.txt
1 23 39	897

input.txt	output.txt
3 1 3 -5 -2 4 1	23

Во втором примере $23 = 3 \cdot 4 + 1 \cdot 1 + (-5) \cdot (-2)$.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def f(first_list: list, second_list: list):
    sum = 0
    first_list.sort()
    second_list.sort()
    for i in range(len(first_list)):
        sum += first_list[i] * second_list[i]
    return sum
```

```
if __name__ == '__main__':
    a = list(map(int, input().split()))
    b = list(map(int, input().split()))
    print(f(a, b))

print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())


'''
Создадим функцию, которая будет отдельно сортировать
прибыль за клик и количество кликов. Далее будем
последовательно
умножать значение прибыли на колиечество кликов и
добавлять полученное произведение в переменную sum.
'''
```

```
Время работы (в секундах): 7.1620702000000005
Память 434, и пик 8825
```

4 задача. Сбор подписей (0.5 балла)

Вы несете ответственность за сбор подписей всех жильцов определенного здания. Для каждого жильца вы знаете период времени, когда он или она находится дома. Вы хотите собрать все подписи, посетив здание как можно меньше раз.

Математическая модель этой задачи следующая. Вам дан набор отрезков на прямой, и ваша цель - отметить как можно меньше точек на прямой так, чтобы каждый отрезок содержал хотя бы одну отмеченную точку.

- **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.

```
import tracemalloc
```

```
import time
t_start = time.perf_counter()
tracemalloc.start()


with open('input1.txt', 'r') as file:
   n=int(file.readline())
   array=[]
   for i in range(n):
        array.append([int(i) for i in
file.readline().split()])
array.sort(key = lambda x: x[1])
votes=[]
while len(array)>0:
    vote=array[0][1]
    votes.append(vote)
    array.pop(0)
    i=0
    while i<len(array):
          if array[i][0] <= vote:
               array.pop(i)
          else:
               i+=1
print(votes)
print(len(votes))
print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())

'''
Мы сортируем массив с помощью встроенной функции,
которая вернет тот элемент, чьё второе значение [1]
больше. Далее мы
уложим в «голоса» промежутки, считанные из файла и в
конце сравним значения списка «голосов» с
соответствующим значением
```

```
промежутка, и, если значение промежутка окажется
меньше - то удалим уго (в противном случае увеличим
i, чтобы продолжить
сравнивать)
'''
```

```
Время работы (в секундах): 0.0002471000000000001
Память 953, и пик 18057
```

8 задача. Расписание лекций (1 балл)

- **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 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 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.
- Примеры:

input.txt	output.txt
1 5 10	1

input.txt	output.txt
3 1 5 2 3 3 4	2

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()

with open('input1.txt', 'r') as file:
    number_of_lec = int(file.readline())
    schedule = [int(i) for i in file.read().split()]
```

```
start = schedule[0::2]
end = schedule[1::2]

prev_time = 0
min_time = 0
num_of_lec_max = 0

while min_time < 1440:
    min_time = 1440
    for i in range(0, number_of_lec):
        if start[i] >= prev_time and end[i] <
min_time:
            min_time = end[i]
    if min_time < 1440:
        prev_time = min_time
        num_of_lec_max += 1


with open('output.txt', 'w') as file:
    file.write(str(num_of_lec_max))


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())

'''
Для этой задачи удобно использовать цикл while.
Создадим два массива, в которых будут храниться время
начала лекций и
время их окончания. Далее, заядя в цикл, находим
минимальное время начала из всех лекций. И если такое
есть,
а максимальное время еще не достигнуто, делаем
найденное минимальное время минимальным для начала
следующей лекции.
```

```
Переменную, в которой хранится количество принятых
заявок, увеличиваем на 1.
'''
```

```
Время работы (в секундах): 0.0006408000000000039
Память 1130, и пик 17660
```

10 задача. Яблоки (1 балл)

- **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.
- Примеры:

input.txt	output.txt
3 5 2 3 10 5 5 10	1 3 2

input.txt	output.txt
3 5 2 3 10 5 5 6	-1

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


with open('input1.txt') as file:
    n, s = map(int, file.readline().split())
    apples = []

    for i in range(n):
        a, b = map(int, file.readline().split())
        apples.append([a, b, i + 1])
```

```
    apples.sort(key=lambda a: [-(a[1] - a[0]), a[0]])
    j = 0
    truth = True
    order = []
    while truth 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
        truth = found
        j += 1
    if truth:
        print(*order)
    else:
        print(-1)


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())

'''
Для этой задачи создадим массив apples, отсортируем
его с помощью лямбда функции. С помощью цикла while
будем смотреть,
можно ли Алисе съесть яблоки так, чтобы ее рост не
стал равным 0. Если можно, выводим порядок номеров
яблок. Если нет,
пишем -1.
'''
```

```
Время работы (в секундах): 0.00022070000000000423
Память 2113, и пик 17910
```

12 задача. Последовательность (1 балл)

- **Постановка задачи.** Дана последовательность натуральных чисел $a_1, a_2, ..., a_n$, 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 $a_1, a_2, ..., a_n$.
- **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 посделовательности 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.
- Примеры:

input.txt	output.txt
3 1 2 3	1 3

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


def func(values: list):
    sum_elem_div_two = sum(values) // 2
    first_set = list()
    second_set = list()
    for value in values:
        if (sum(first_set) + value) <=
sum_elem_div_two:
            first_set.append(value)
        else:
            second_set.append(value)
```

```
    if sum(first_set) == sum(second_set) ==
sum_elem_div_two:
        return str(second_set.__len__()) + "\n" +
second_set.__str__()
    else:
        return -1


if __name__ == '__main__':
    list_of_value = list(map(int, input().split()))
    print(func(list_of_value))


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())

'''
Узнаем половину суммы всех множеств. Создадим два
подмножества. В цикле for заполним их при условии,
что сумма каждого
из этих подмножеств не больше половины суммы всех
подмножеств. И если в конечном итоге подмножества
равны, то возвращаем
количество элементов в любом (в данном случае втором)
подмножестве и номера элементов в этом подмножестве.
Иначе -
возвращаем  -1.
'''
 Время работы (в секундах): 5.0350918
 Память 552, и пик 8701
```

14 задача. Максимальное значение арифметического выражения (2 балла)

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
1+5	6

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

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

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


file1=open('output.txt', 'w')
file2=open('input1.txt', 'r')
str1 = file2.readline()
if not (len(str1) >= 0 and len(str1) <= 14):
   raise ValueError
numbers = []
operators=[]
```

```
m=[]
M=[]
for i in str1:
    if i=="+" or i=="-" or i=="*":
        operators.append(i)
    else:
        numbers.append(int(i))


def minimax(i, j, m, M, operators):
    min1 = float("+inf")
    max1 = float("-inf")
    for k in range(i, j):
        if operators[k] == '*':
            a = M[i][k] * M[k + 1][j]
            b = M[i][k] * m[k + 1][j]
            c = m[i][k] * M[k + 1][j]
            d = m[i][k] * m[k+1][j]
        elif operators[k] == '+':
            a = M[i][k] + M[k + 1][j]
            b = M[i][k] + m[k + 1][j]
            c = m[i][k] + M[k + 1][j]
            d = m[i][k] + m[k + 1][j]
        else:
            a = M[i][k] - M[k + 1][j]
            b = M[i][k] - m[k + 1][j]
            c = m[i][k] - M[k + 1][j]
            d = m[i][k] - m[k + 1][j]
        min1 = min(min1, a, b, c, d)
        max1 = max(max1, a, b, c, d)
    return min1, max1


def maxvalue(numbers, operators):
    n = len(numbers)
    m = []
    M = []
    for i in range(len(numbers)):
```

```
        m.append([])
        M.append([])
        for j in range(len(numbers)):
            m[i].append(0)
            M[i].append(0)
    for i in range(n):
        m[i][i] = numbers[i]
        M[i][i] = numbers[i]
    for s in range(1, n):
        for i in range(n-s):
            j = i+s
            m[i][j], M[i][j] = minimax(i, j, m, M,
operators)
    return M[0][n - 1]


sum = maxvalue(numbers, operators)
file1.write(str(sum))
file1.close()


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())


'''
В функции minimax находим минимальное и максимальное
значение для каждого оператора. В функции maxvalue
заполняем
массивы m и M и ищем максимальный результат при
выставлении скобок.
'''
 Время работы (в секундах): 0.0006449999999999997
 Память 12052, и пик 26859
```

20 задача. Почти палиндром (3 балла)

- **Постановка задачи.** Слово называется палиндромом, если его первая буква совпадает с последней, вторая – с предпоследней и т.д. Например: «abba», «madam», «x».

  Для заданного числа K слово называется почти палиндромом, если в нем можно изменить не более K любых букв так, чтобы получился палиндром. Например, при $K = 2$ слова «*reactor*», «*kolobok*», «*madam*» являются почти палиндромами (подчеркнуты буквы, заменой которых можно получить палиндром).

  Подсловом данного слова являются все слова, получающиеся путем вычеркивания из данного нескольких (возможно, одной или нуля) первых букв и нескольких последних. Например, подсловами слова «cat» являются слова «c», «a», «t», «ca», «at» и само слово «cat» (а «ct» подсловом слова «cat» не является).

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

- **Формат входного файла (input.txt).** В первой строке входного файла вводятся два натуральных числа: N – длина слова и K. Во второй строке записано слово S, состоящее из N строчных английских букв.

- **Ограничения на входные данные.** $1 \leq N \leq 5000, 0 \leq K \leq N$.

- **Формат выходного файла (output.txt).** В выходной файл требуется вывести одно число – количество подслов слова S, являющихся почти палиндромами (для данного K).

- Ограничение по времени. 2 сек.

- Ограничение по памяти. 256 мб.

- Примеры:

input.txt	output.txt	input.txt	output.txt
5 1 abcde	12	3 3 aaa	6

- Проверить можно по ссылке.

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()

def palindrome(string, k):
    n = len(string)
```

```
    j = -1
    x = 0
    for i in range(n//2):
        if string[i] != string[j]:
            x += 1
            if x > k:
                return False
        j -= 1
    return True

def subwords(word):
    allswords = []
    for i in range(len(word)):
        for n in range(i+1, len(word)+1):
            allswords.append(word[i:n])
    return allswords

if __name__=='__main__':
    with open('input1.txt') as file:
        ln = file.readline()
        k = int(ln.split()[-1])
        word = file.readline()

    swords = subwords(word)
    count = 0
    for word in swords:
        if palindrome(word, k):
            count += 1
    print(count)
    with open('output.txt', 'wt') as file:
        file.write(str(count))


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())
```

```
'''
Runtime error :(
Создадим функцию, которая будет сравнивать слово по
буквам с левого и правого концов. Также создадим
вторую функцию,
в которой будет массив с хранящимися в нем
подсловами. И если в ходе решения выясняется, что
подслово - почти палиндром,
переменную count увеличиваем на 1.
'''
```

```
Время работы (в секундах): 0.0006478000000000039
Память 2106, и пик 18114
```

21 задача. Игра в дурака (3 балла)

- **Постановка задачи.** Петя очень любит программировать. Недавно он решил реализовать популярную карточную игру «Дурак». Но у Пети пока маловато опыта, ему срочно нужна Ваша помощь.

  Как известно, в «Дурака» играют колодой из 36 карт. В Петиной программе каждая карта представляется в виде строки из двух символов, где первый символ означает ранг (‘6’, ‘7’, ‘8’, ‘9’, ‘T’, ‘J’, ‘Q’, ‘K’, ‘A’) карты, а второй символ означает масть (‘S’, ‘C’, ‘D’, ‘H’). **Ранги перечислены в порядке возрастания старшинства.**

  Пете необходимо решить следующую задачу: сможет ли игрок, обладая набором из N карт, отбить M карт, которыми под него сделан ход? Для того чтобы отбиться, игроку нужно покрыть каждую из карт, которыми под него сделан ход, картой из своей колоды. Карту можно покрыть либо старшей картой той же масти, либо картой козырной масти. Если кроющаяся карта сама является козырной, то её можно покрыть только старшим козырем. Одной картой можно покрыть только одну карту.

- **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

```
import tracemalloc
import time
t_start = time.perf_counter()
tracemalloc.start()


with open("input1.txt") as file:
    N, M, R = map(str, file.readline().split())
```

```
    my = list(map(str, file.readline().split()))
    enemy = list(map(str, file.readline().split()))

actions = {'6': ['7', '8', '9', '10', 'J', 'Q', 'K',
'A'],
           '7': ['8', '9', '10', 'J', 'Q', 'K', 'A'],
           '8': ['9', '10', 'J', 'Q', 'K', 'A'],
           '9': ['10', 'J', 'Q', 'K', 'A'],
           '10': ['J', 'Q', 'K', 'A'],
           'J': ['Q', 'K', 'A'],
           'Q': ['K', 'A'],
           'K': ['A'],
           'A': [None],
           'trump': ['6', '7', '8', '9', 'T', 'J',
'Q', 'K', 'A']}

N, M = int(N), int(M)
S = []
C = []
D = []
H = []
for i in range(N):
    if my[i][1] == 'S':
        S.append(my[i][0])
    elif my[i][1] == 'C':
        C.append(my[i][0])
    elif my[i][1] == 'D':
        D.append(my[i][0])
    else:
        H.append(my[i][0])
for i in range(M):
    beaten = False
    suit = enemy[i][1]
    card = enemy[i][0]
    if suit == 'S':
        for j in actions[card]:
            if beaten == False:
                if j in S:
```

```
                S.remove(j)
                beaten = True
elif suit == 'C':
    for j in actions[card]:
        if beaten == False:
            if j in C:
                C.remove(j)
                beaten = True
elif suit == 'D':
    for j in actions[card]:
        if beaten == False:
            if j in D:
                D.remove(j)
                beaten = True
else:
    for j in actions[card]:
        if beaten == False:
            if j in H:
                H.remove(j)
                beaten = True
if beaten==False:
    if suit==R:
        with open('output.txt', 'w') as file:
            file.write('NO')
        exit()
    else:
        if R=='S':
            for j in actions['trump']:
                if beaten == False:
                    if j in S:
                        S.remove(j)
                        beaten = True
        if R=='C':
            for j in actions['trump']:
                if beaten == False:
                    if j in C:
                        C.remove(j)
                        beaten = True
```

```
            if R=='D':
                for j in actions['trump']:
                    if beaten == False:
                        if j in D:
                            D.remove(j)
                            beaten = True
            else:
                    for j in actions['trump']:
                        if beaten == False:
                            if j in H:
                                H.remove(j)
                                beaten = True
            if beaten==False:
                with open('output.txt', 'w') as file:
                    file.write('NO')
                exit()
with open('output.txt', 'w') as file:
    file.write('YES')


print("Время работы (в секундах):",
time.perf_counter()-t_start)
print("Память %d, и пик %d" %
tracemalloc.get_traced_memory())

'''
Runtime error:(
Создадим словарь, в котором укажем, какие карты
(значения) какую могут крыть (ключ). Далее, создадим
4 массива,
соответствующие 4 мастям в колоде карт. В цикле for
проверяем, можем ли мы отбить карту. Если нет,
выводим в ответ False,
если да - Yes.
'''
```

```
Время работы (в секундах): 0.0006714000000000026
Память 3842, и пик 18236
```

Вывод:

В данной лабораторной работе мы вновь рассмотрели и применили на практики техники «жадного» и динамического программирований. Кроме того, в работе мы вновь применили методы работы с массивами, встроенные функции и сортировки, а так же другие возможности программирования на языке Python.