

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 27

Выполнил:  
Филиппов А.Э.  
К3139

Проверила:  
Артамонова В.Е.

Санкт-Петербург

2023 г.

## Содержание отчета

<b>Содержание отчета</b>	<b>3</b>
<b>Задачи по варианту</b>	<b>4-21</b>
Задача №4. Сбор подписей [0.5 баллов]	4-7
Задача №5. Максимальное количество призов [0.5 баллов]	8-10
Задача №9. Распечатка [1 балл]	11-14
Задача №19. Произведение матриц [3 балла]	14-17
Задача №20. Почти палиндром [3 балла]	17-21
<b>Дополнительные задачи</b>	<b>22-</b>
Задача №1. Максимальная стоимость добычи [0.5 баллов]	22-24
Задача №2. Заправки [0.5 баллов]	25-28
Задача №3. Максимальный доход от рекламы [0.5 баллов]	28-31
Задача №6. Максимальная зарплата [0.5 баллов]	31-35
Задача №7. Проблема сапожника [0.5 баллов]	35-38
Задача №8. Расписание лекций [1 балл]	38-41
Задача №10. Яблоки [1 балл]	41-44
Задача №11. Максимальное количество золота [1 балл]	45-47
Задача №12. Последовательность [1 балл]	47-50
Задача №13. Сувениры [1.5 балла]	51-54
Задача №14. Максимальное значение арифметического выражения [2 балла]	54-57
Задача №15. Удаление скобок [2 балла]	57-60
Задача №16. Продавец [2 балла]	61-65
Задача №17. Ход конем [2.5 балла]	65-68
Задача №18. Кафе [2.5 балла]	69-72
Задача №21. Игра в дурака [3 балла]	73-76
Задача №22. Симпатичные узоры [4 балла]	76-81
<b>Вывод</b>	<b>82</b>

## Задачи по варианту

### Задача №4. Сбор подписей [0.5 баллов]

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

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

- **Постановка задачи.** Дан набор из  $n$  отрезков  $[a_0, b_0], [a_1, b_1], \dots, [a_{n-1}, b_{n-1}]$  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.
- Примеры:

№	input.txt
1	3 1 3 2 5 3 6

№	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 time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
output = open("output.txt", "w")

n = int(lines[0])
wohner = []
for i in range(n):
    start, end = map(int, lines[i+1].split())
    wohner.append([start, end])
wohner = sorted(wohner, key=lambda x: x[0])
minstart = None
minend = None
out = []
```

```

count = 0
for start, end in wohner:
    if minstart == None:
        minstart = start
        minend = end
    else:
        if start > minend:
            out.append(minend)
            minstart = start
            minend = end
            count += 1
        else:
            minend = min(minend, end)
            minstart = max(minstart, start)

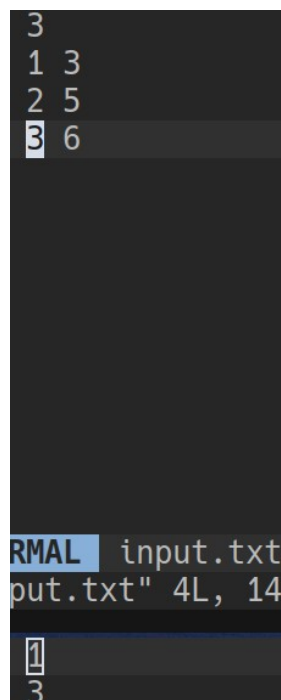
output.write(str(count+1) + "\n")
output.write("".join(map(str, out + [minend])))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Сначала данные сортируются по началу отрезка. Далее проходимся циклом по списку, запоминая минимальный конец отрезка. Если начало отрезка больше чем минимальный конец отрезка, то добавляем координату в ответ, добавляем единицу к счетчику.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



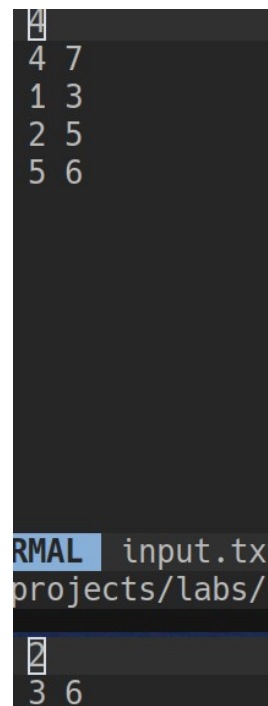
```

3
1 3
2 5
3 6

RMAL input.txt
put.txt" 4L, 14

1
3

```



```

4
4 7
1 3
2 5
5 6

RMAL input.tx
projects/labs/

2
3 6

```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

```

1
1 3
858073225 904297000
480801680 812339760
658538428 983255813
585333080 843728733
470192731 710762348
223391597 808056516
471156615 753477931
626359975 785636578
124590290 138908312
669626220 769539355
947213080 995462868
40589469 298064667
201121718 684597187
RMAL input.txt
out.txt" 101L, 1996B
text utf-8[unix] 201 words 0% ln :1/101≡:1
4
138908312 179568585 311527848 424162354 621607100 654829395 726865849 765244065 776672829 863236885 886877183 908919639 97
4965083 992884607
RMAL input.txt
out.txt" 2L, 6B
1
3

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00017095200018957257	13941
Пример из задачи	0.00019258800057286862	14047
Пример из задачи	0.00023765200057823677	14100
Верхняя граница диапазона значений входных данных из текста задачи	0.0006371649997163331	33797

Вывод по задаче: Жадные алгоритмы используются в самых разных задачах. Эта задача смогла наглядно показать, что жадный подход может потребоваться в самых неожиданных задачах.

### Задача №5. Максимальное количество баллов [0.5 баллов]

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.**  $1 \leq n \leq 109$ .
- **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.
- **Примеры:**

№	input.txt	output.txt	№	input.txt	output.txt
1	6	3 1 2 3	2	8	3 1 2 5

№	input.txt	output.txt
3	2	1 2



```

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

file = open('input.txt')
lines = file.readlines()
output = open("output.txt", "w")

n = int(lines[0])
summ = 0
i = 1
out = []
while summ + i <= n:
    summ += i
    out.append(i)
    i += 1
if summ != n:
    out[len(out)-1] += (n - summ)
output.write(str(len(out)) + "\n")
output.write(" ".join(map(str, out)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Подбираем последовательность 1 2 3 ... пока сумма не будет больше или равна n. Если сумма больше n, то вычитаем последнее число и формируем новое.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```

6
RMAL input.txt
projects/labs/sem
3
1 2 3

```

```

2
RMAL inp
out.txt"
1
2

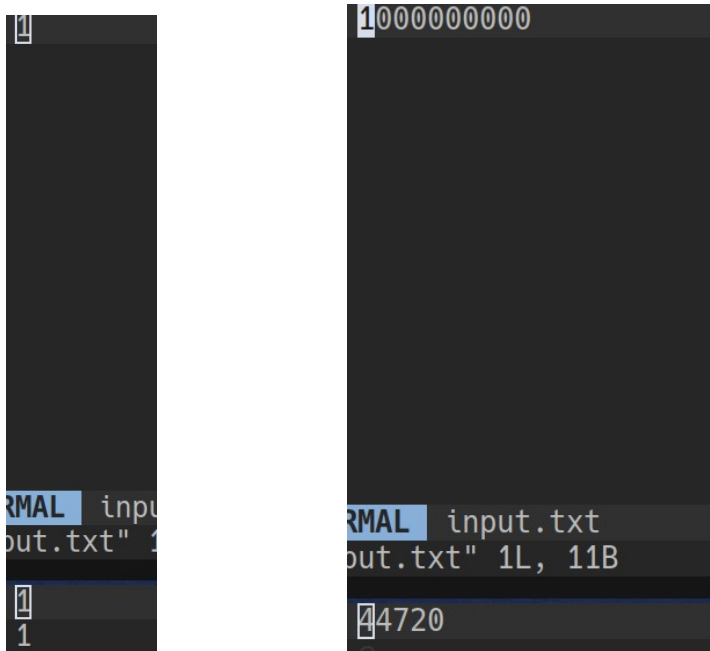
```

```

8
RMAL input.
put.txt" 1L,
3
1 2 5

```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00015460699978575576	13888
Пример из задачи	0.00015446300039911876	13888
Пример из задачи	0.00020378099998197285	13888
Пример из задачи	0.00014758799989067484	13888
Верхняя граница диапазона значений входных данных из текста задачи	0.07617324100010592	4706559

Вывод по задаче: Решение данной задачи помогло закрепить практику решения алгоритмов «жадным» приемом. Данная задача классический пример жадного алгоритма

### Задача №9. Распечатка [1 балл]

• **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  $A_1$  рублей, 10 листов - за  $A_2$  рублей, 100 листов - за  $A_3$  рублей, 1000 листов - за  $A_4$  рублей, 10000 листов - за  $A_5$  рублей, 100000 листов - за  $A_6$  рублей, 1000000 листов - за  $A_7$  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, застав 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 по дается 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.
- Примеры:

input.txt	output.txt	input.txt	output.txt
980	882	980	900
1		1	
9		10	
90		100	
900		1000	
1000		900	
10000		10000	
10000		10000	

```

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

file = open('input.txt')
lines = file.readlines()
output = open("output.txt", "w")

n = int(lines[0])
a = []
for i in range(7):
    cost = int(lines[i+1])
    cost_pro_blatt = cost / (10 ** i)
    a.append({"blaetter": 10**i, "cost": cost,
              "cost_pro_blatt": cost_pro_blatt})
a = sorted(a, key=lambda x: x["cost_pro_blatt"])
cost = 0
mini = 9999999999999999999
n_ = n
for i in range(7):
    order = a[i]
    if order["blaetter"] <= n:
        count = (n // order["blaetter"])
        n -= order["blaetter"] * count
        cost += order["cost"] * count
    if (order["cost"] < mini) and (order["blaetter"] >= n_):
        mini = order["cost"]

output.write(str(min(cost, mini)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Создаем массив для каждого числа бумаги. Вычисляем для каждого числа стоимость за одну бумагу, из этого делаем жадное решение  
 Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```
980
1
9
90
900
1000
10000
10000
NORMAL input.txt
projects/labs
882
```

```
980
1
10
100
1000
900
10000
10000
NORMAL input.txt
out.txt" 8L,
900
```

Результат работы кода на максимальных и минимальных значениях:  
 (скрины input output файлов)

```
1
1
10
100
1000
900
10000
10000
NORMAL input.txt
put.txt" 8L,
1
```

```
20000000000
1
10
100
1000
900
10000
10000
NORMAL input.txt
200000000
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00020731899894599337	14721
Пример из задачи	0.00024456699975416996	14763
Пример из задачи	0.00020731899894599337	14729
Верхняя граница диапазона значений входных данных из текста задачи	0.0002069469992420636	14748

Вывод по задаче: Решение данной задачи помогло закрепить практику решения задач динамического программирования. Данная задача классический пример задачи дп.

### Задача №19. Произведение матриц [3 балла]

- **Постановка задачи.** В произведении последовательности матриц полно стью расставлены скобки, если выполняется один из следующих пунктов:

- Произведение состоит из одной матрицы.
- Оно является заключенным в скобки произведением двух произведений с полностью расставленными скобками.

Полная расстановка скобок называется оптимальной, если количество операций, требуемых для вычисления произведения, минимально.

Требуется найти оптимальную расстановку скобок в произведении последовательности матриц.

- **Формат ввода / входного файла (input.txt).** В первой строке входных данных содержится целое число  $n$  - количество матриц. В  $n$  следующих строк содержится по два целых числа  $a_i$  и  $b_i$  -

количество строк и столбцов в  $i$ -той матрице соответственно. Гарантируется, что  $bi = ai + 1$  для любого  $1 \leq i \leq n - 1$ .

- **Ограничения на входные данные.**  $1 \leq n \leq 400$ ,  $1 \leq ai, bi \leq 100$  для всех  $1 \leq i \leq n$ .

- **Формат вывода / выходного файла (output.txt).** Выведите оптимальную расстановку скобок. Если таких расстановок несколько, выведите любую.

- Ограничение по времени. 2 сек.

- Ограничение по памяти. 64 мб.

- Пример:

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

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

```
from math import inf
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
output = open("output.txt", "w")

n = int(lines[0]) + 1
groesse = [0] * n
dp = [[inf for j in range(n)] for i in range(n)]
ant = [[0 for j in range(n)] for i in range(n)]
```

```

def calc(l, r):
    if dp[l][r] == inf:
        for i in range(l, r):
            curr = calc(l, i) + calc(i+1, r) + \
                (groesse[l-1] * groesse[i] * groesse[r])
            if curr < dp[l][r]:
                dp[l][r] = curr
                ant[l][r] = i

    return dp[l][r]

def antBauen(l, r):
    if l == r:
        return 'A'

    return '(' + antBauen(l, ant[l][r]) + antBauen(ant[l][r] + 1, r) + ')'

for i in range(1, n):
    groesse[i - 1], groesse[i] = list(map(int, lines[i].split()))
    if dp[i][i]:
        dp[i][i] = 0

calc(1, n-1)
output.write(str(antBauen(1, n-1)))
print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Это классическая задача динамического программирования. Решается динамическим программированием

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```

3
10 50
50 90
90 20
[ ]

MAL input
out.txt" 5
[(AA)A)

```



Результат работы кода на максимальных и минимальных значениях:  
 \_\_\_\_\_(скрины input output файлов)

[illegible]

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00021226299986665254	13943
Пример из задачи	0.0002378989997851022	15042
Верхняя граница диапазона значений входных данных из текста задачи	0.2751263469999685	395222

Вывод по задаче: Решение данной задачи помогло закрепить практику решения задач динамического программирования. Данная задача классический пример задачи дп.

### Задача №20. 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», «х».

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 «с», «а», «t», «са», «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](#).

```

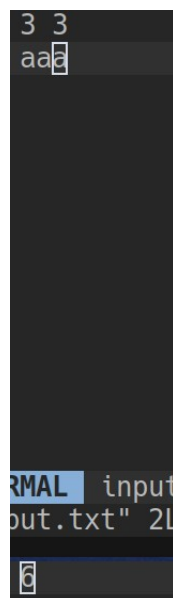
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
output = open("output.txt", "w")

n, k = map(int, lines[0].split())
dp = [[-1 for i in range(n)] for j in range(2)]
s = lines[1]
count = 1
for i in range(n-1):
    if s[i] == s[i+1]:
        dp[1][i] = 0
        count += 1
    else:
        dp[1][i] = 1
        if 1 <= k:
            count += 1
    dp[0][i] = 0
    count += 1
dp[0][n-1] = 0
for L in range(3, n+1):
    dp_new = [-1 for i in range(n)]
    for i in range(n-L+1):
        j = i + L - 1
        index = (L-1) % 2
        if s[i] != s[j]:
            dp_new[i] = dp[index][i+1] + 1
        else:
            dp_new[i] = dp[index][i+1]
        if dp_new[i] <= k:
            count += 1
    dp[index % 2] = dp_new
output.write(str(count))
print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

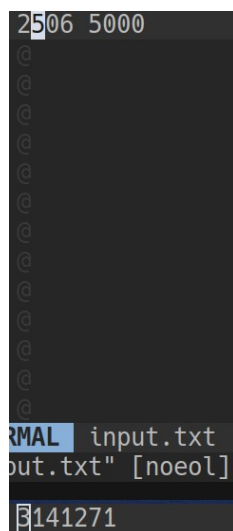
```

Классическая задача динамического программирования. Для того, чтобы задача проходила по памяти применен следующий метод оптимизации: сохраняем только 2 последних состояния динамики.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



```
1 0
1
MAL i
ut.txt
1
```



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001654619999840179	13941
Пример из задачи	0.00022523799998452887	13945
Пример из задачи	0.0001623669999730737	13943

Верхняя граница диапазона значений входных данных из текста задачи	7.05187446900004	240012
---	------------------	--------

Вывод по задаче: Для оптимального решения задачи иногда нужно правильно подобрать баланс между расходом памяти и затратами по времени. Для решения данной задачи, чтобы удовлетворить ограничения по памяти пришлось увеличить время выполнения программы.

## Дополнительные задачи

### Задача №1. Максимальная стоимость добычи [0.5 баллов]

Вор находит гораздо больше добычи, чем может поместиться в его сумке. Помогите ему найти самую ценную комбинацию предметов, предполагая, что любая часть предмета добычи может быть помещена в его сумку. Цель - реализовать алгоритм для задачи о дробном рюкзаке.

- **Формат ввода / входного файла (input.txt).** В первой строке входных данных задано целое число  $n$  - количество предметов, и  $W$  - вместимость сумки. Следующие  $n$  строк определяют значения веса и стоимости предметов. В  $i$ -ой строке содержатся целые числа  $p_i$  и  $w_i$  — стоимость и вес  $i$ -го предмета, соответственно.
- **Ограничения на входные данные.**  $1 \leq n \leq 103$ ,  $0 \leq W \leq 2 \cdot 10^6$ ,  $0 \leq p_i \leq 2 \cdot 10^6$ ,  $0 \leq w_i \leq 2 \cdot 10^6$  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.
- 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

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

```

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

n, w = map(int, lines[0].split())

sachen = []
for i in range(n):
    preis, gewicht = map(int, lines[i+1].split())
    preis_pro_gewicht = preis / gewicht
    sachen.append([gewicht, preis_pro_gewicht])
sachen = sorted(sachen, key=lambda x: x[1], reverse=True)
antwort = 0
for gewicht, ppg in sachen:
    minusgewicht = min(gewicht, w)
    w -= minusgewicht
    antwort += minusgewicht * ppg
    if w == 0:
        break
out.write("{0:.4f}".format(antwort))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Сначала предметы сортируются по соотношению цена/вес. Далее ответ формируется исходя из сортированного списка

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```

3 50
1 60 20
2 100 50
3 120 30

```

ORMAL input.txt

180.0000

```

1 10
500 30

```

ORMAL input.txt  
out.txt" 2L, 12B

66.6667

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

```
1 10
500 30

RMAL input.txt
out.txt" 2L, 12B

66.6667
```

```
100 200000
12845 35481
25189 99175
72090 83026
180826 88291
59641 132484
97576 65
131612 148853
191692 135524
25981 9196
192090 108956
123670 32723
129463 180141
68611 198744
RMAL input.txt
out.txt" 101L, 128

579292.9100
```

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

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00025277900022047106	13947
Пример из задачи	0.00019135399998049252	14058
Пример из задачи	0.00025277900022047106	13947
Верхняя граница диапазона значений входных данных из текста задачи	0.0005803750000268337	32210

Вывод по задаче: Жадным методом можно решать различный спектр задач. Тип данной задачи — дробный рюкзак



## Задача №2. Заправки [0.5 баллов]

Вы собираетесь поехать в другой город, расположенный в  $d$  км от вашего родного города. Ваш автомобиль может проехать не более  $m$  км на полном баке, и вы начинаете с полным баком. По пути есть заправочные станции на расстояниях  $stop1, stop2, \dots, stopn$  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 строке 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.
- 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.t xt	output.t xt	input.t xt	output.t xt
10		200	
3		250	
4	-1	2	0
1 2 5		100	
9		150	

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

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

d = int(lines[0])
m = int(lines[1])
n = int(lines[2])
stops = list(map(int, lines[3].split()))
effective_spot = 0
count = 0
last = 0
for i in range(n):
    stop = stops[i]
    if stop - effective_spot <= m:
        last = stop
    else:
        count += 1
        effective_spot = last
if d - effective_spot <= m:
    out.write(str(count))
else:
    if d - last <= m:
```

```

        out.write(str(count+1))
    else:
        out.write("-1")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Данную задачу можно решить действуя «жадно». А именно: ехать до дальнейшей доступной заправки, пока это возможно.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

The first screenshot shows input.txt with values 950, 400, 4, 200, 375, 550, 750 and output.txt with 4L, 26B. The second screenshot shows input.txt with 10, 3, 4, 1, 2, 5, 9 and output.txt with 4L, 15B. The third screenshot shows input.txt with 200, 250, 2, 100, 150 and output.txt with 4L, 18B. Each screenshot has a terminal window with a dark background and a light blue prompt.

Результат работы кода на максимальных и минимальных значениях: (скрины input output файлов)

The first screenshot shows input.txt with 1, 1, 1, 2 and output.txt with 4L, 15B. The second screenshot shows input.txt with 100000, 400, 300 and output.txt with 4L, 18B. Each screenshot has a terminal window with a dark background and a light blue prompt.

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

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00013588500041805673	14042
Пример из задачи	0.008553000000574684	14059
Пример из задачи	0.00016291399970214115	14048
Пример из задачи	0.00019448100010777125	14051
Верхняя граница диапазона значений входных данных из текста задачи	0.000641151000309037	42179

Вывод по задаче: Жадным методом можно решать различный спектр задач. В том числе такие произвольные типы задач.

### Задача №3. Максимальный доход от рекламы [0.5 баллов]

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.
- **Примеры:**

input.txt	output.txt	input.txt	output.txt
1 23 39	897	3 1 3 -5 -2 4 1	23

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

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

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

n = int(lines[0])
a = list(map(int, lines[1].split()))
b = list(map(int, lines[2].split()))
negative_a = []
negative_b = []
positive_a = []
positive_b = []
for i in range(n):
    if a[i] <= 0:
        negative_a.append(a[i])
    else:
        positive_a.append(a[i])
    if b[i] <= 0:
        negative_b.append(b[i])
    else:
        positive_b.append(b[i])
negative_a = sorted(negative_a)
negative_b = sorted(negative_b)

summ = 0
if len(negative_b) > len(negative_a):
```

```

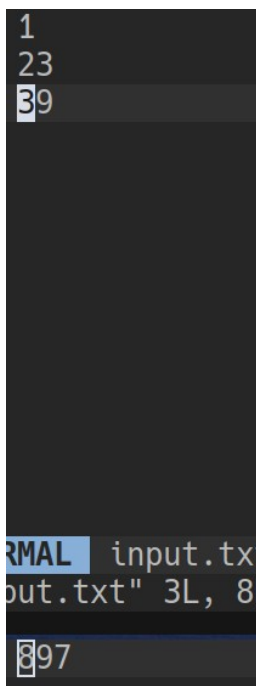
    for i in range(len(negative_a)):
        summ += negative_a[i] * negative_b[i]
    for i in range(len(negative_a), len(negative_b)):
        positive_b.append(negative_b[i])
else:
    for i in range(len(negative_b)):
        summ += negative_a[i] * negative_b[i]
    for i in range(len(negative_b), len(negative_a)):
        positive_a.append(negative_a[i])

positive_a = sorted(positive_a)
positive_b = sorted(positive_b)
for i in range(len(positive_a)):
    summ += positive_a[i] * positive_b[i]
out.write(str(summ))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Формируется список положительных элементов и другой список остальных элементов. Исходя из них формируется итоговый ответ. Результат работы кода на примерах из текста задачи:(скрины input output файлов)



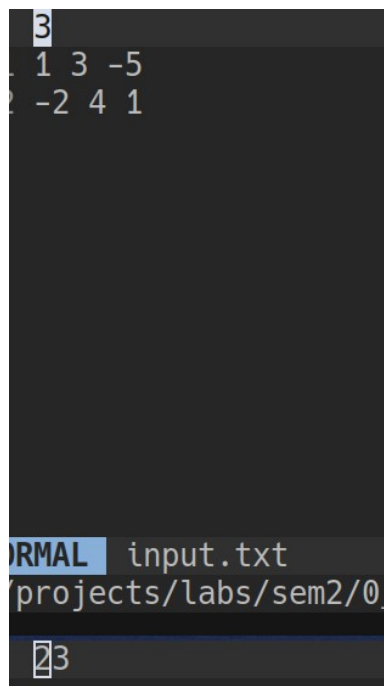
```

1
23
39

```

ORMAL input.txt  
out.txt" 3L, 81

97



```

3
1 3 -5
-2 4 1

```

ORMAL input.txt  
projects/labs/sem2/0

23

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)



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  $a_1, a_2, \dots, a_n$ .
- **Ограничения на входные данные.**  $1 \leq n \leq 102$ ,  $1 \leq a_i \leq 103$  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.



• Пример:

input.txt	output.txt	input.txt	output.txt
2 21 2	221	3 23 39 92	923923

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

```
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")
```

```
def swap(i, j):
    d = a[i]
    a[i] = a[j]
    a[j] = d
```

```
def cmp(a, b):
    if len(a) > len(b):
        mini = len(b)
        first = int(a[:mini])
        second = int(b)
        if first > second:
            return True
        else:
            return False
    elif len(b) > len(a):
        mini = len(a)
        first = int(a)
        second = int(b[:mini])
        if second > first:
            return False
        else:
            return True
    else:
        if int(a) > int(b):
            return True
        else:
            return False
```

```
def quick_sort(l, r):
    i = l
    j = r
    pivot = nums[(i+j)//2]
```

```

while i <= j:
    while cmp(nums[i], pivot):
        i += 1
    while cmp(pivot, nums[j]) and (pivot != nums[j]):
        j -= 1
    if i <= j:
        nums[i], nums[j] = nums[j], nums[i]
        i += 1
        j -= 1
if j > l:
    quick_sort(l, j)
if i < r:
    quick_sort(i, r)

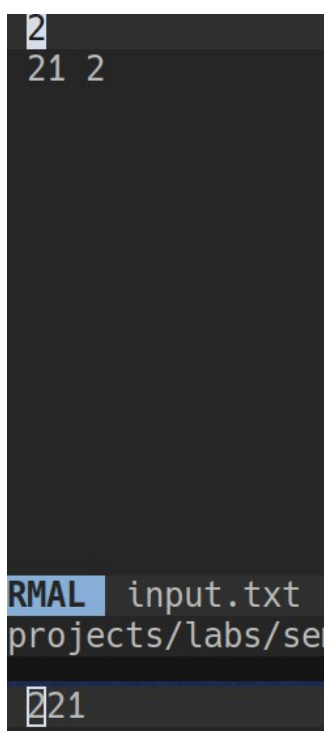
n = int(lines[0])
nums = list(map(str, lines[1].split()))
quick_sort(0, n-1)
out.write("".join(nums))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Для решения данной задачи используется quick sort. Единственным отличием является функция сравнения — сравниваются только части до минимальной длины двух сравниваемых чисел.

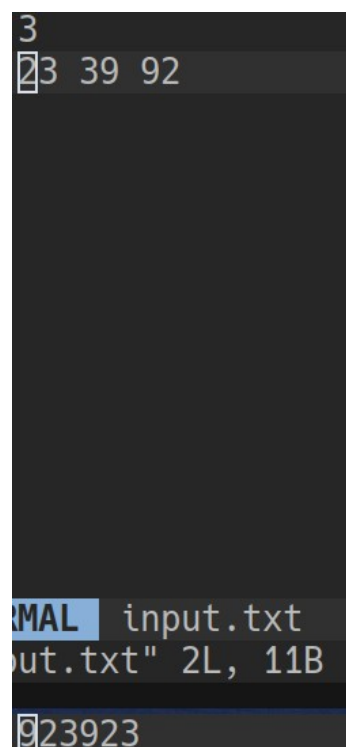
Результат работы кода на примерах из текста задачи:(скрины input output файлов)



2  
21 2

RMAL input.txt  
projects/labs/se

21



3  
23 39 92

RMAL input.txt  
out.txt" 2L, 11B

23923

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00019291900025564246	13939
Пример из задачи	0.00016615799995634006	13942
Пример из задачи	0.00016411000069638249	13946
Верхняя граница диапазона значений входных данных из текста задачи	0.001545672999782255	24129

Вывод по задаче: Жадные алгоритмы бывают разными. Они могут включать в себя изменения алгоритмов сортировки

### Задача №7. Проблема сапожника [0.5 баллов]

• **Постановка задачи.** В некоей воинской части есть сапожник. Рабочий день сапожника длится  $K$  минут. Заведующий складом оценивает работу

са пожника 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.
- Примеры:

input.txt	output.txt
10 3 6 2 8	2

input.txt	output.txt
3 2 10 20	0

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

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

k, n = map(int, lines[0].split())
```

```

stiefeln = sorted(list(map(int, lines[1].split())))
summ = 0
count = 0
for i in range(n):
    if summ + stiefeln[i] <= k:
        summ += stiefeln[i]
        count += 1
    else:
        break
out.write(str(count))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Сперва сортируем массив. Далее берем самые минимальные по времени сапоги пока это возможно.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```

10 3
6 2 8

RMAL input.txt
projects/labs/sem
2

```

```

3 2
10 20

RMAL input.txt
out.txt" 2L, 10B
0

```

Результат работы кода на максимальных и минимальных значениях: (скрины input output файлов)

```

1 1
1

RMAL input.txt
out.txt" 2L,
1

```

```

1000 500
91 69 33 28 82 68 95 49 5 24 78 58 16 80 62 45 20 20
8 27 55 8 36 51 19 17 31 95 10 63 66 44 23 61 98 62 5
43 88 56 72 33 45 15 69 40 72 93 68 37 11 89 70 100
54 41 33 20 26 84 47 19 66 29 20 2 45 87 20 69 63 21
0 95 6 62 64 97 90 19 60 13 90 8 91 61 24 19 87 40 39
65 15 52 17 59 78 1 13 81 55 58 54 5 91 34 25 42 11 4
74 76 65 38 18 84 3 36 23 46 21 56 3 80 52 56 43 88
86 38 46 75 68 75 83 63 48 7 3 16 100 58 45 12 17 96
41 42 38 64 98 32 2 74 53 24 9 45 11 9 53 62 20 25 89
19 6 18 24 73 41 91 19 69 64 51 45 32 83 18 56 2 51
22 51 40 6 16 54 34 18 64 35 34 62 62 67 5 1 89 100 9
94 11 9 43 36 51 51 24 68 73 53 67 79 72 6 65 16 8 3

RMAL input.txt
out.txt" [noeol] 2L, 1463B
8

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00015916899974399712	13941
Пример из задачи	0.008361924999917392	13946
Пример из задачи	0.00019890600015060045	13945
Верхняя граница диапазона значений входных данных из текста задачи	0.0006887429999551387	43543

Вывод по задаче: Решение данной задачи помогло закрепить практику решения алгоритмов «жадным» приемом. Данная задача классический пример жадного алгоритма

#### **Задача №8. Расписание лекции [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  $[ si, fi )$  - время начала и конца лекции. Лекция считается открытым интервалом, то есть какая-то лекция может начаться в момент окончания другой, без перерыва. Необходимо выбрать из этих заявок такое подмножество, чтобы суммарно выполнить максимальное количество заявок. Учтите, что одновременно в лекционной аудитории, конечно же, может читаться лишь одна лекция.

- **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  $si$  и  $fi$  для каждой лекции  $i$ . Гарантируется, что  $si < fi$ . Время начала и окончания лекции - натуральные числа, не превышают 1440 (в минутах с начала суток).

- **Ограничения на входные данные.**  $1 \leq N \leq 1000$ ,  $1 \leq si, fi \leq 1440$
- **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
3 1 5 2 3 3 4	2

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

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

n = int(lines[0])
vorlesungen = []
for i in range(n):
    start, end = map(int, lines[i+1].split())
    vorlesungen.append([start, end])
vorlesungen = sorted(vorlesungen, key=lambda x: x[1])
persist_end = 0
count = 0
for start, end in vorlesungen:
    if start >= persist_end:
        count += 1
```

```

        persist_end = end
out.write(str(count))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Задача оказалась проще чем та, что на 0.5 баллов. Нужно делать все то же самое: отсортировать, пройтись циклом с учетом запоминания минимального конца лекции.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```

1
5 10

MAL input.txt
out.txt" 2L, 7B
1

```

```

3
1 5
2 3
3 4

MAL input.txt
out.txt" 4L, 14B
2

```

Результат работы кода на максимальных и минимальных значениях: (скрины input output файлов)

```

1
5 10

MAL input.txt
out.txt" 2L, 7B
1

```

```

1000
101 402
1146 1170
586 900
1162 1237
1168 1282
1260 1419
818 986
255 1181
423 1333
583 1413
530 603
535 544
1054 1210
MAL input.txt
out.txt" 1001L, 8
60

```



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00016514700018888107	13942
Пример из задачи	0.0001650359999985085	14047
Пример из задачи	0.00016514700018888107	13942
Верхняя граница диапазона значений входных данных из текста задачи	0.004530604000137828	232490

Вывод по задаче: Решение данной задачи помогло закрепить практику решения алгоритмов «жадным» приемом. Данная задача классический пример жадного алгоритма

#### Задача №10. 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 со всем. 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 по рядке, в котором их нужно есть.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

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

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

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

n, s = map(int, lines[0].split())
aepfel = []

for i in range(n):
    verloren, bekommen = map(int, lines[i+1].split())
    aepfel.append([i, verloren, bekommen])
aepfel = sorted(aepfel, key=lambda x: x[2] - x[1], reverse=True)

count = 0
curr = s
fehler = False
ans = []
hist = [False] * n

while count < n:
    gefunden = False
    for i in range(n):
        apfel = aepfel[i]
        if (curr > apfel[1]) and (not hist[apfel[0]]):
            count += 1
            gefunden = True
            curr += (bekommen - verloren)
            if curr < 0:
                fehler = True
                break
    if not gefunden:
        ans.append(-1)
        break
    else:
        ans.append(count)
        hist[apfel[0]] = True
```

```

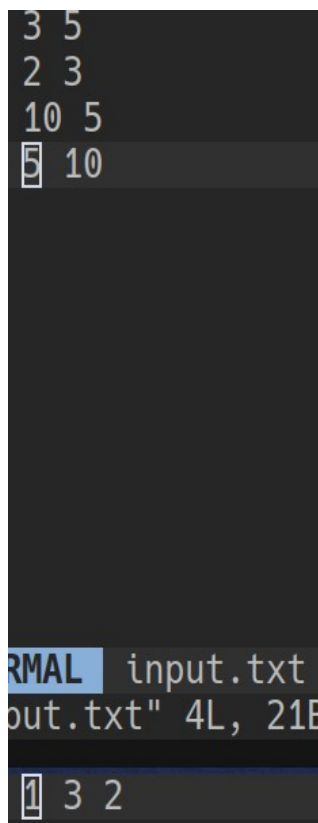
        out.write(str(-1))
        fehler = True
        hist[apfel[0]] = True
        ans.append(apfel[0] + 1)
        break
    if not gefunden:
        fehler = True
        out.write(str(-1))
        break
    if fehler:
        break
if not fehler:
    out.write(" ".join(map(str, ans)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Отсортируем яблоки по выгоде. Будем есть самые выгодные яблоки и исключать съеденные из списка, пока не съедим все или не сможем их больше есть.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



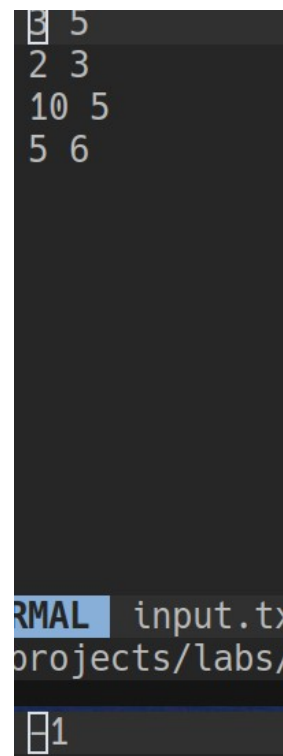
```

3 5
2 3
10 5
5 10

RMAL input.txt
out.txt" 4L, 21B

1 3 2

```



```

3 5
2 3
10 5
5 6

RMAL input.tx
projects/labs/

1

```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

```
1 1
1 1
...
RMAL input.
out.txt" 2L,
1
```

```
1000 791
1 253 332
2 231 215
3 239 366
4 199 869
5 27 236
6 801 880
7 662 388
8 880 84
9 298 533
0 552 782
1 216 328
2 924 994
3 951 108
RMAL input.txt
out.txt" 1001L, 7792
1
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001835349994507851	13943
Пример из задачи	0.008458262998829014	14050
Пример из задачи	0.00024214000040956307	14054
Верхняя граница диапазона значений входных данных из текста задачи	0.004392190001453855	273233

Вывод по задаче: Решение данной задачи помогло закрепить практику решения алгоритмов «жадным» приемом. Данная задача классический пример жадного алгоритма

### Задача №11. Максимальное количество золота [1 балл]

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

- **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.
- Пример:

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

Здесь сумма весов первого и последнего слитка равна 9.

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

```
import time
import tracemalloc
tracemalloc.start()
```

```

t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

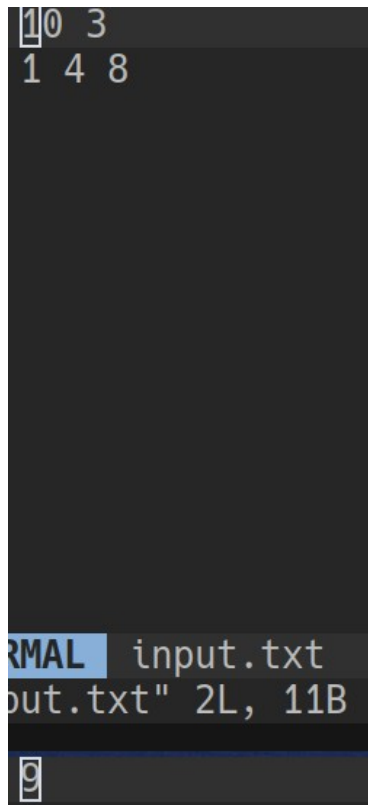
w, n = map(int, lines[0].split())
gewichte = list(map(int, lines[1].split()))
dp = [[0 for x in range(w+1)] for y in range(n+1)]
for i in range(1, n+1):
    for j in range(1, w+1):
        if j >= gewichte[i-1]:
            dp[i][j] = max(dp[i-1][j], dp[i-1]
                           [j - gewichte[i-1]] + gewichte[i-1])
        else:
            dp[i][j] = dp[i-1][j]
out.write(str(dp[n][w]))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Здесь классический пример задачи о рюкзаке. Решается посредством динамического программирования

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



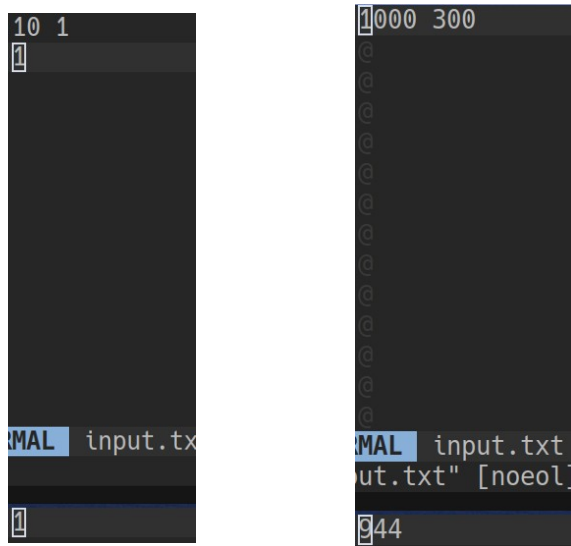
```

10 3
1 4 8

NORMAL input.txt
out.txt" 2L, 11B

```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00019269399854238145	13942
Пример из задачи	0.008318366000821698	13946
Верхняя граница диапазона значений входных данных из текста задачи	0.18306449300143868	2693042

Вывод по задаче: Данная задача является задачей недробного рюкзака. Такие задачи можно решать при помощи динамического программирования

### Задача №12. Последовательность [1 балл]

• **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.
- Примеры:

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

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

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

n = int(lines[0])
nums = list(map(int, lines[1].split()))

s = sum(nums) // 2
output = []

if sum(nums) % 2 != 0:
    out.write(str(-1))
else:
    dp = [[0 for i in range(s+1)] for y in range(n)]
    for i in range(n):
        dp[i][0] = 1
```



```

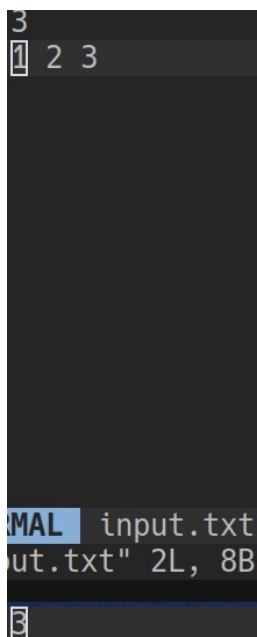
dp[0][nums[0]] = 1
for i in range(1, n):
    for j in range(1, s+1):
        dp[i][j] = dp[i-1][j]
        if nums[i] <= j:
            dp[i][j] = max(dp[i][j], dp[i-1][j-nums[i]])
if dp[n-1][s] != 1:
    out.write(str(-1))
else:
    i = n-1
    j = s
    while j != 0 and i > 0:
        if dp[i-1][j-nums[i]] == 0:
            i -= 1
        else:
            output.append(nums[i])
            j -= nums[i]
            i -= 1
    if j != 0:
        output.append(nums[0])

    output.reverse()
    out.write("".join(map(str, output)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Задача о рюкзаке опять же. Цель динамики — получить из чисел сумму, деленную на два. Как базу можно обозначить данные нам числа, далее просто заполнять двумерный массив динамики, по нему строим ответ. Результат работы кода на примерах из текста задачи:(скрины input output файлов)



```

3
2 3

MAL input.txt
ut.txt" 2L, 8B

```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001713399997242959 2	13940
Пример из задачи	0.0002013180001085857	13943
Верхняя граница диапазона значений входных данных из текста задачи	0.03306718999920122	4127402

Вывод по задаче: Благодаря этой задаче я научился решать задачи известные как subset sum problems. Это классические задачи динамического программирования, которые могут использоваться в биоинформатике.

### Задача №13. Сувениры [1.5 балла]

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

- **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.
- Примеры:

input.txt	output.txt	input.txt	output.txt
4 3 3 3 3	0	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 time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

def obMoeglich(n, a, b, c):
    if a == 0 and b == 0 and c == 0:
        return True
    if n < 0:
        return False

    if a >= nums[n]:
        if obMoeglich(n-1, a - nums[n], b, c):
            return True

    if b >= nums[n]:
        if obMoeglich(n-1, a, b - nums[n], c):
            return True

    if c >= nums[n]:
        if obMoeglich(n-1, a, b, c - nums[n]):
            return True

    return False

n = int(lines[0])
nums = list(map(int, lines[1].split()))
s = sum(nums) // 3
if sum(nums) % 3 != 0:
    out.write("0")
else:
    out.write(str(int(obMoeglich(n-1, s, s, s))))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Данное решение выполнено перебором. Поскольку у задачи не такие строгие рамки

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```
1
17 59 34 57 17 23 67 1 18 2 59

RMAL input.txt
projects/labs/sem2/0_alg/Дополните
```

```
1
0

RMAL input.
out.txt" 2L,
```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

```
1
0

RMAL input.
out.txt" 2L,
```

```
0
8 5 7 5 2 7 27 4 16 27 27 11 13 23 8 1 20 29 15 24

RMAL input.txt
out.txt" [noeol] 2L, 54B
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00016147899987117853	13940
Пример из задачи	0.0003191150008206023	15676
Пример из задачи	0.00016147899987117853	13940
Верхняя граница диапазона значений входных данных из текста задачи	0.0003358349986228859	

Вывод по задаче: Динамика не всегда может являться очевидным решением задачи. Иногда перебора вполне достаточно

#### Задача №14. 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  $s_0, s_1, \dots, s_{2n}$ . Каждый символ в четной позиции  $s$  является цифрой (то есть целым числом от 0 до 9), а каждый символ в нечетной позиции является одной из трех операций из  $+, -, *$
- **Ограничения на входные данные.**  $0 \leq n \leq 14$  (следовательно, строка содержит не более 29 символов).
- **Формат вывода / выходного файла (output.txt).** Выведите максимально возможное значение заданного арифметического

выражения среди различных порядков применения арифметических операций.

- Ограничение по времени. 5 сек.

- Пример:

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

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

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

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

def operate(num1, op, num2):
    if op == "-":
        return num1 - num2
    elif op == "+":
        return num1 + num2
    else:
        return num1 * num2

def dp_max(start, end):
    maxi = -999999999
    if start == end:
        return nums[start]
    for k in range(start, end):
        val = max(operate(dp_max(start, k), ops[k], dp_max(k+1, end)),
                  operate(dp_max(start, k), ops[k], dp_min(k+1, end)),
                  operate(dp_min(start, k), ops[k], dp_max(k+1, end)),
                  operate(dp_min(start, k), ops[k], dp_min(k+1, end)))
        if val > maxi:
            maxi = val
    return maxi

def dp_min(start, end):
    mini = 999999999999
```

```

    if start == end:
        return nums[start]
    for k in range(start, end):
        val = min(operate(dp_max(start, k), ops[k], dp_max(k+1, end)),
                  operate(dp_max(start, k), ops[k], dp_min(k+1, end)),
                  operate(dp_min(start, k), ops[k], dp_max(k+1, end)),
                  operate(dp_min(start, k), ops[k], dp_min(k+1, end)))
        if val < mini:
            mini = val
    return mini

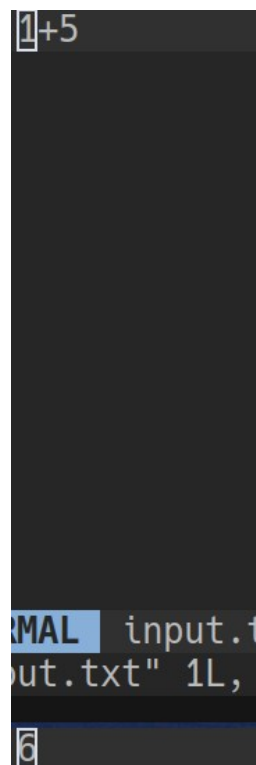
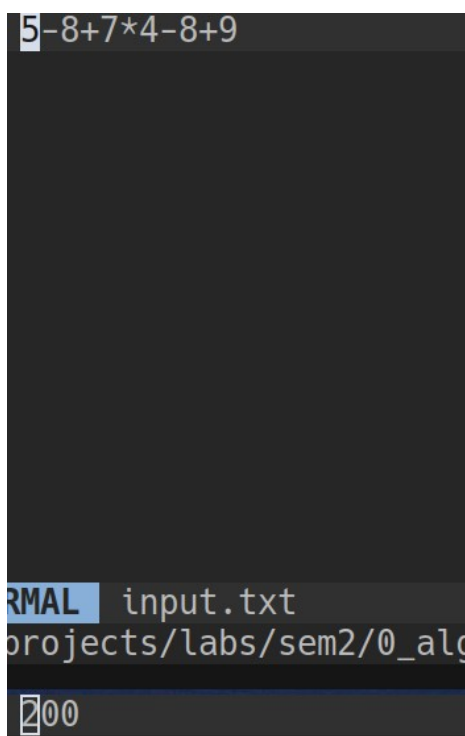
string = lines[0]
nums = []
ops = []
for i in range(len(string)):
    if i % 2 == 0:
        nums.append(int(string[i]))
    else:
        ops.append(string[i])
out.write(str(dp_max(0, len(nums)-1)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

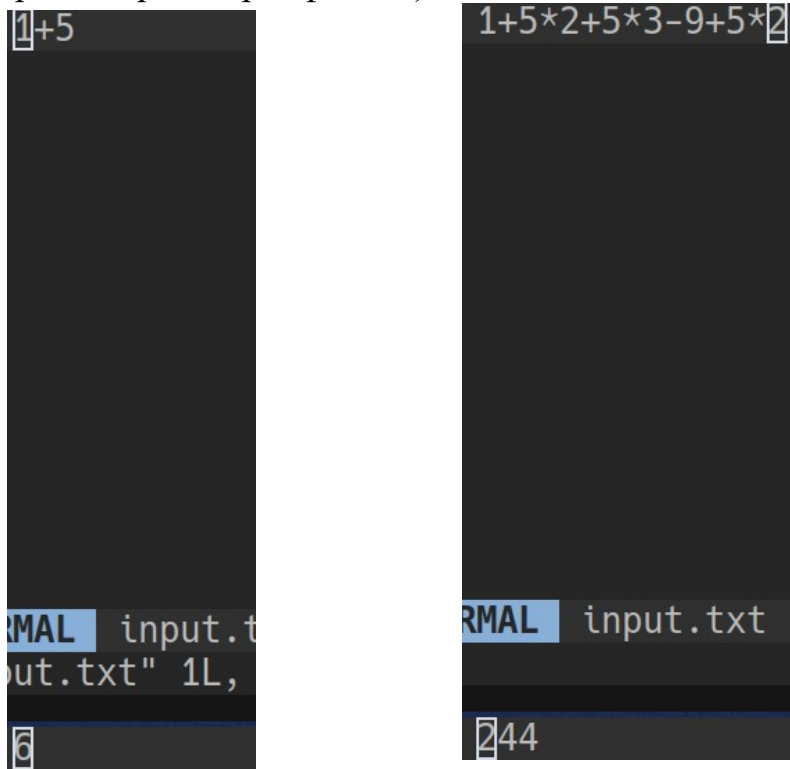
Из всех вариантов с операциями вычитания, сложения и умножения перебираются минимальные и максимальные. Исходя из этого строится максимальный ответ.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)





Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001667989999987185	13890
Пример из задачи	0.025014120999912848	17071
Пример из задачи	0.0001667989999987185	13890
Верхняя граница диапазона значений входных данных из текста задачи	1.8626964039995073	18287

Вывод по задаче: Динамика не всегда может являться очевидным решением задачи. Иногда перебора вполне достаточно

### Задача №15. 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 time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

def ans(l, r):
    if best[l][r] == -1:
        res[l] = res[r] = ''
    if l == r:
        return
    if best[l][r] == -2:
        if l + 1 == r:
            return
        ans(l+1, r-1)
    else:
        ans(l, best[l][r])
        ans(best[l][r] + 1, r)
    return
```

```

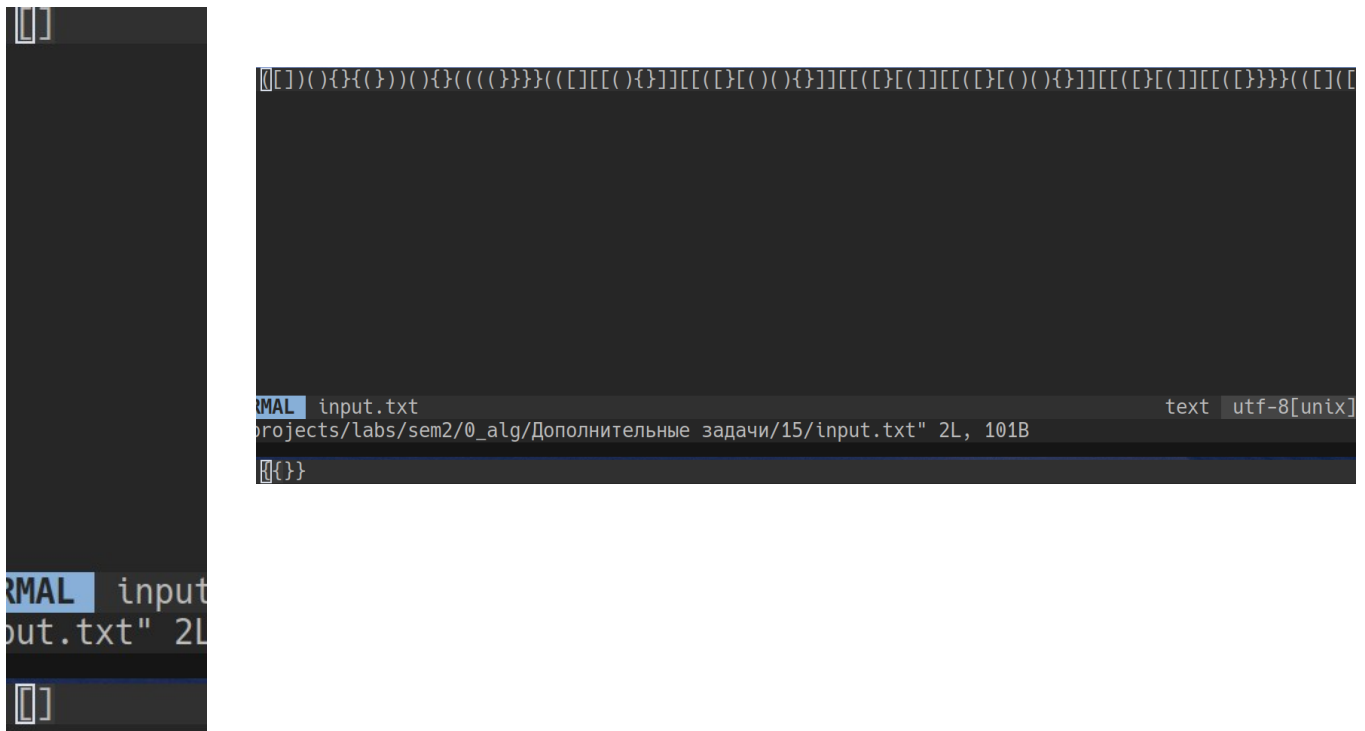
inf = 10 ** 9
s = lines[0].strip()
n = len(s)
dp = [[inf] * (n+1) for i in range(n+1)]
best = [[-1] * (n+1) for i in range(n+1)]
for i in range(n+1):
    dp[i][i] = 1
for i in range(n-1):
    if (s[i] == '(' and s[i+1] == ')') or (s[i] == '[' and s[i+1] == ']')
or (s[i] == '{' and s[i+1] == '}'):
        dp[i][i+1] = 0
        best[i][i+1] = -2
for length in range(2, n + 1):
    for l in range(n - length + 1):
        r = l + length - 1
        if (s[l] == '(' and s[r] == ')') or (s[l] == '[' and s[r] == ']')
or (s[l] == '{' and s[r] == '}'):
            if dp[l][r] > dp[l+1][r-1]:
                dp[l][r] = dp[l+1][r-1]
                best[l][r] = -2
            for m in range(l, r):
                curr = dp[l][m] + dp[m+1][r]
                if dp[l][r] > curr:
                    dp[l][r] = curr
                    best[l][r] = m
res = list(s)
ans(0, n-1)
out.write(''.join(res))
print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Данная задача решается динамически. Базой динамики будут строки длиной 1 и длиной 2. Те, что образуют правильную скобочную последовательность будут равны единице в двумерном массиве динамики. Из этого можно строить дальнейшее решение. Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0001716800015856279	13889
Пример из задачи	0.00018235599964100402	13891
Верхняя граница диапазона значений входных данных из текста задачи	0.04907070300032501	225672

Вывод по задаче: Классическая задача на динамику. Решение данной задачи помогло закрепить практику решения задач динамическим программированием

### Задача №16. Продавец [2 балла]

• **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 –  $ai,j$  – это расстояние между городами  $i$  и  $j$ .
- **Ограничения на входные данные.**  $1 \leq n \leq 11$ ,  $0 \leq ai,j \leq 106$ ,  $ai,j = aj,i$ ,  $ai,i = 0$ .
- **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. 3 сек.
- 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
-----------

```
#include <bits/stdc++.h>

using namespace std;

#define ll long long

void process_mem_usage(double& vm_usage, double& resident_set)
{
    vm_usage      = 0.0;
    resident_set  = 0.0;

    // the two fields we want
    unsigned long vsize;
    long rss;
    {
        std::string ignore;
        std::ifstream ifs("/proc/self/stat", std::ios_base::in);
        ifs >> ignore >> ignore >> ignore >> ignore >> ignore >> ignore >>
ignore >> ignore >> ignore >> ignore
        >> ignore >> ignore >> ignore >> ignore >> ignore >> ignore
>> ignore >> ignore >> ignore >> ignore
        >> ignore >> ignore >> vsize >> rss;
    }

    long page_size_kb = sysconf(_SC_PAGE_SIZE) / 1024; // in case x86-64 is
configured to use 2MB pages
    vm_usage = vsize / 1024.0;
    resident_set = rss * page_size_kb;
}

ll n;
ll dist[1048576][20];

ll visited_all;

typedef struct {
    vector<ll> visited;
    ll ant = LLONG_MAX;
} ans;

vector<ll> visited;

ans tsp(ll mask, ll pos) {
    if (mask == visited_all) {
        return {visited, dist[pos][0]};
    }
    ans mem;
    ll ant = LLONG_MAX;
    for (int stadt = 0; stadt < n; stadt++) {
```

```

        if ((mask & (1 << stadt)) == 0) {
            visited.push_back(stadt);
            ans res = tsp(mask | (1 << stadt), stadt);
            res.ant += dist[pos][stadt];
            if (res.ant < ant) {
                mem = res;
                ant = mem.ant;
            }
            visited.pop_back();
        }
    }
    return mem;
}

int main() {
    ifstream input; input.open("input.txt");
    ofstream output; output.open("output.txt");
    auto start = chrono::high_resolution_clock::now();
    double vm, rss;
    process_mem_usage(vm, rss);
    input >> n;
    for (ll i = 0; i < n; i++) {
        for (ll j = 0; j < n; j++)
            input >> dist[i][j];
    }
    visited_all = (1 << n) - 1;
    ll ant = LLONG_MAX;
    ll mask = 1;
    ans mem;
    int mem_stadt;
    for (int stadt = 0; stadt < n; stadt++) {
        if ((mask & (1 << stadt)) == 0) {
            ans res = tsp(mask | (1 << stadt), stadt);
            if (res.ant < ant) {
                mem = res;
                ant = mem.ant;
                mem_stadt = stadt;
            }
        }
    }

    output << mem.ant << "\n" << mem_stadt+1 << " ";
    for (ll i = 0; i < mem.visited.size(); i++)
        output << mem.visited[i] + 1 << " ";

    auto end = chrono::high_resolution_clock::now();

    double diff = chrono::duration<double, std::milli>(end-start).count();
    cout << "Время выполнения: " << diff << endl;
}

```

```

    cout << "Использовано памяти: " << vm;
    return 0;
}

```

Для решения данной задачи используется динамическое программирование. Прием, использованный в данной задаче называется динамическое программирование на маске, маску используют, чтобы сэкономить и время и память. В нашем случае маской будем обозначать посещенные города.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```

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

RMAL input.txt
out.txt" 7L, 94B

66
4 5 2 3 1

```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

```

1
1

RMAL in

```

```

1
0 183 163 173 181 183 163 173 181 183 173
183 0 165 172 171 183 163 173 181 183 173
163 165 0 189 302 183 163 173 181 183 173
173 172 189 0 167 183 163 173 181 183 173
181 171 302 167 0 183 163 173 181 183 173
181 171 302 167 0 183 163 173 181 183 173
181 171 302 167 0 183 163 173 181 183 173
181 171 302 167 0 183 163 173 181 183 173
181 171 302 167 0 183 163 173 181 183 173
181 171 302 167 0 183 163 173 181 183 173
181 171 302 167 0 183 163 173 181 183 173

RMAL input.txt
out.txt" 13L, 467B

539
6 4 7 5 8 9 10 11 2 3 1

```



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

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00167177	169676
Пример из задачи	0.00271094	169676
Верхняя граница диапазона значений входных данных из текста задачи	2.76332	169676

Вывод по задаче: С битовыми операциями намного проще работать в C++ чем в питоне. Поэтому для некоторых задач резонно использовать C++.

### Задача №17. Ход конем [2.5 балла]

• **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.

1 2 3
4 5 6
7 8 9 .
0 .

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.
- Примеры:

input.txt	output.txt	input.txt	output.txt
1	8	2	16

```

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

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

n = int(lines[0]) - 1
arr = {
    "0": 0,
    "1": 1,
    "2": 1,
    "3": 1,
    "4": 1,
    "5": 1,
    "6": 1,
    "7": 1,
    "8": 0,
    "9": 1
}
arr_new = {
    "0": 0,
    "1": 0,
    "2": 0,
    "3": 0,
    "4": 0,
    "5": 0,
    "6": 0,
    "7": 0,
    "8": 0,
    "9": 0
}
for i in range(n):

```

```

arr_new["0"] = arr["4"] + arr["6"]
arr_new["1"] = arr["8"] + arr["6"]
arr_new["2"] = arr["7"] + arr["9"]
arr_new["3"] = arr["8"] + arr["4"]
arr_new["4"] = arr["0"] + arr["9"] + arr["3"]
arr_new["6"] = arr["0"] + arr["7"] + arr["1"]
arr_new["7"] = arr["6"] + arr["2"]
arr_new["8"] = arr["3"] + arr["1"]
arr_new["9"] = arr["4"] + arr["2"]
arr = copy.deepcopy(arr_new)
arr_new = {
    "0": 0,
    "1": 0,
    "2": 0,
    "3": 0,
    "4": 0,
    "5": 0,
    "6": 0,
    "7": 0,
    "8": 0,
    "9": 0
}
summ = 0
for key in arr_new:
    summ += arr[key]
out.write(str(summ))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

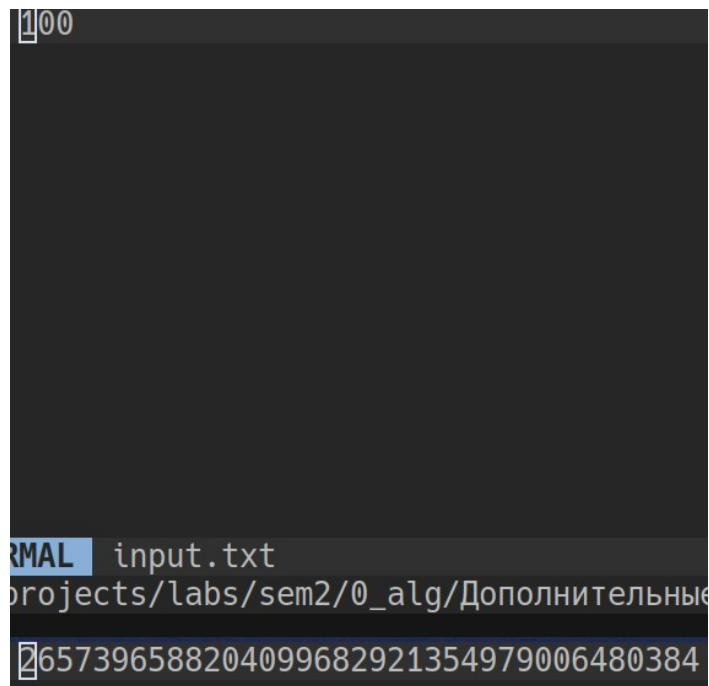
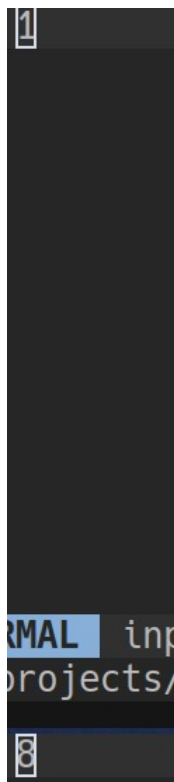
```

Задача решается динамическим программированием. База динамики задается условиями задачи, а именно: все клетки кроме 0 и 8. Далее создается динамика согласно условию задачи.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)



	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00015565599824185483	13888
Пример из задачи	0.0001538379983685445	13888
Пример из задачи	0.00019405300190555863	14294
Верхняя граница диапазона значений входных данных из текста задачи	0.0026159629996982403	15397

Вывод по задаче: Классическая задача на динамику. Решение данной задачи помогло закрепить практику решения задач динамическим программированием

### Задача №18. Кафе [2.5 балла]

- **Постановка задачи.** Около университета недавно открылось новое кафе, в котором действует следующая система скидок: при каждой покупке бо лее 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 минималь но 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 возраста ющем порядке номера дней, когда вам следует воспользоваться купонами. Если существует несколько решений с минимальной суммарной стоимо стью, 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
3	220
110	1 1
110	2
110	

```

from math import inf
import copy
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

def poor_student(n):
    matrix = [[0 for j in range(n + 1)] for i in range(n + 1)]
    mem = [[[0, 0, []] for j in range(n + 1)] for i in range(n + 1)]
    for i in range(1, n + 1):
        matrix[0][i] = inf
    for i in range(1, n + 1):
        if matrix[i - 1][1] > matrix[i - 1][0] + prices[i] and (
            matrix[i - 1][1] == inf or prices[i] <= 100
        ):
            matrix[i][0] = matrix[i - 1][0] + prices[i]
            mem[i][0] = copy.deepcopy(mem[i - 1][0])
            if prices[i] > 100:
                mem[i][0][0] += 1
        else:
            matrix[i][0] = matrix[i - 1][1]
            mem[i][0] = copy.deepcopy(mem[i - 1][1])
            mem[i][0][1] += 1
            mem[i][0][2].append(i)
    for j in range(1, n):
        if prices[j] <= 100:
            price1 = matrix[i - 1][j] + prices[j]
            price2 = matrix[i - 1][j + 1]
            if price1 < price2:
                matrix[i][j] = price1
                mem[i][j] = copy.deepcopy(mem[i - 1][j])

```

```

        else:
            matrix[i][j] = price2
            mem[i][j] = copy.deepcopy(mem[i-1][j+1])
    else:
        price1 = matrix[i-1][j-1] + prices[i]
        price2 = matrix[i-1][j+1]
        if price1 < price2:
            matrix[i][j] = price1
            mem[i][j] = copy.deepcopy(mem[i-1][j-1])
            mem[i][j][0] += 1
        else:
            matrix[i][j] = price2
            mem[i][j] = copy.deepcopy(mem[i-1][j+1])
            mem[i][j][1] += 1
            mem[i][j][2].append(i)
    out.write(str(matrix[n][0]))
    out.write("\n" + str(mem[n][0][0] - mem[n]
        [0][1]) + " " + str(mem[n][0][1]))

n = int(lines[0])
prices = [0]
for i in range(n):
    prices.append(int(lines[i+1]))
poor_student(n)

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
    str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Данная задача — немного переделанная задача из разряда динамического программирования из раздела «рюкзак». Решается аналогичным способом. Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```

3
110
110
110

MAL input
projects/l

220
1 1

```

```

5
110
40
120
110
60

MAL in
out.txt"

60
0 2

```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

```
0
MAL input
out.txt" 1
0 0
```

```
100
146
229
37
267
11
75
4
108
94
83
54
84
292
MAL input
549
0 31
```

18866667	21.02.2023 19:07:59	Филиппов Артём Эдуардович	0268	PyPy	Accepted	0,75	8,4 Мб
----------	---------------------	---------------------------	------	------	----------	------	--------

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00015581000002384826	13552
Пример из задачи	0.0003413039999031753	18182
Пример из задачи	0.0004984979998425842	22415
Верхняя граница диапазона значений входных данных из текста задачи	0.19920923400013635	3195054

Вывод по задаче: Классическая задача на динамику. Решение данной задачи помогло закрепить практику решения задач динамическим программированием



### Задача №21. 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 на ходятся два натуральных числа  $N$  и  $M$ , а также символ  $R$ , означающий козырную масть. Во второй строке перечислены  $N$  карт, находящихся на руках у игрока. В третьей строке перечислены  $M$  карт, которые необходимо отбить. Все карты отделены друг от друга одним пробелом.
- **Ограничения на входные данные.**  $N \leq 35$ ,  $M \leq 4$ ,  $M \leq N$ .
- **Формат выходного файла (output.txt).** В выходной файл выведите «YES» в случае, если отбиться можно, либо «NO», если нельзя.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 16 мб.
- Примеры:

input.txt	output.txt	input.txt	output.txt
-----------	------------	-----------	------------

6 2 C KD KC AD 7C AH 9C 6D 6C	YES	4 1 D 9S KC AH 7D 8D	NO
-------------------------------------	-----	-------------------------------	----

- Проверить можно по [ссылке](#).

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

```
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")
```

```
def karteZuZahl(num):
    wechseIn = {
        "6": 0,
        "7": 1,
        "8": 2,
        "9": 3,
        "T": 4,
        "J": 5,
        "Q": 6,
        "K": 7,
        "A": 8
    }
    return wechseIn[num]
```

```
sortierte_karte = {"S": [], "C": [], "D": [], "H": []}
```

```
def greedSort():
    for i in range(len(karte)):
        el = karte[i]
        sortierte_karte[el[1]].append(
            karteZuZahl(el[0])
        )
    sortierte_karte["S"].sort()
    sortierte_karte["C"].sort()
    sortierte_karte["D"].sort()
    sortierte_karte["H"].sort()
```

```
n, m, r = lines[0].split()
```

```

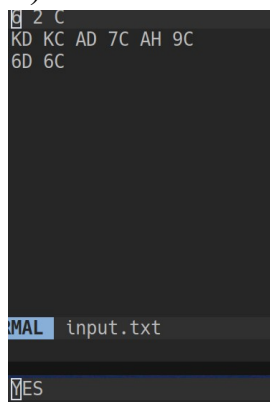
n = int(n)
m = int(m)
karte = lines[1].split()
greedSort()
schlagen = lines[2].split()
ans = True
for i in range(len(schlagen)):
    el = schlagen[i]
    familie = el[1]
    nummer = karteZuZahl(el[0])
    istSchlagbar = False
    for j in range(len(sortierte_karte[familie])):
        el2 = sortierte_karte[familie][j]
        if el2 > nummer:
            istSchlagbar = True
            sortierte_karte[familie].pop(j)
            break
    if not istSchlagbar and familie != r:
        for j in range(len(sortierte_karte[r])):
            sortierte_karte[r].pop(j)
            istSchlagbar = True
            break
    if not istSchlagbar:
        ans = False
        break
if ans:
    out.write("YES")
else:
    out.write("NO")

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Создается ассоциативный массив с мастями, где карты разных мастей отсортированы. Далее жадно имитируется игра: сначала пытаемся отбить наименьшей возможной картой той же масти, если не получается тянемся за козырем. Если карту не получается отбить, то выводим ответ нет, иначе да.

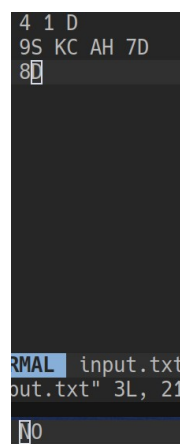
Результат работы кода на примерах из текста задачи:(скрины input output файлов)



```

4 2 C
KD KC AD 7C AH 9C
6D 6C
MAL input.txt
YES

```



```

4 1 D
9S KC AH 7D
8
MAL input.txt
out.txt" 3L, 21
NO

```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

18869910	22.02.2023 11:11:49	Филиппов Артём Эдуардович	0698	PyPy	Accepted		0,203	428 Кб
----------	---------------------	---------------------------	------	------	----------	--	-------	--------

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.000183538000101179 93	13996
Пример из задачи	0.000168093999945995 17	14014
Пример из задачи	0.0001899680000086082 6	14005
Верхняя граница диапазона значений входных данных из текста задачи	0.000232675000006565 82	15685

Вывод по задаче: Решение данной задачи помогло закрепить практику решения алгоритмов «жадным» приемом. Данная задача классический пример жадного алгоритма

### Задача №22. Симпатичные узоры [4 балла]

- **Постановка задачи.** Компания BrokenTiles планирует заняться выкладыва нием во дворах у состоятельных клиентов узор из

черных и белых плиток, каждая из которых имеет размер  $1 \times 1$  метр. Известно, что дворы всех состоятельных людей имеют наиболее модную на сегодня форму прямоугольника  $M \times N$  метров.

Однако при составлении финансового плана у директора этой организации появилось целых две серьезные проблемы: во первых, каждый новый клиент очевидно захочет, чтобы узор, выложенный у него во дворе, отличался от узоров всех остальных клиентов этой фирмы, а во вторых, этот узор должен быть симпатичным. Как показало исследование, узор является **симпатичным**, если в нем нигде не встречается квадрата  $2 \times 2$  метра, полностью покрытого плитками одного цвета.

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

- **Формат входного файла (input.txt).** В первой строке входного файла на ходятся два положительных целых числа, разделенные пробелом –  $M$  и  $N$ .
- **Ограничения на входные данные.**  $1 \leq N \times M \leq 30$ .
- **Формат выходного файла (output.txt).** Выведите в выходной файл единственное число – количество различных симпатичных узоров, которые можно выложить во дворе размера  $M \times N$ . Узоры, получающиеся друг из друга сдвигом, поворотом или отражением считаются различными.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 16 мб.
- Примеры:

input.txt	output.txt	input.txt	output.txt
xt	xt	xt	xt
2 2	14	3 3	322

- Проверить можно по [ссылке](#).

```

#include <bits/stdc++.h>

using namespace std;

#define ll long long

void process_mem_usage(double& vm_usage, double& resident_set)
{
    vm_usage      = 0.0;
    resident_set = 0.0;

    // the two fields we want
    unsigned long vsize;
    long rss;
    {
        std::string ignore;
        std::ifstream ifs("/proc/self/stat", std::ios_base::in);
        ifs >> ignore >> ignore >> ignore >> ignore >> ignore >> ignore >> ignore >>
ignore >> ignore >> ignore >> ignore
        >> ignore >> ignore >> ignore >> ignore >> ignore >> ignore
>> ignore >> ignore >> ignore >> ignore
        >> ignore >> ignore >> vsize >> rss;
    }

    long page_size_kb = sysconf(_SC_PAGE_SIZE) / 1024; // in case x86-64 is
configured to use 2MB pages
    vm_usage = vsize / 1024.0;
    resident_set = rss * page_size_kb;
}

const int maxi = 1 << 6;

int n,m;

int dp[maxi][maxi];
int a[31][maxi];

bool calc(int i, int j) {
    bool b[4];
    for (int i_ = 0; i_ < n - 1; i_++) {
        b[0] = (i & (1 << i_)) != 0;
        b[1] = (i & (1 << (i_ + 1))) != 0;
        b[2] = (j & (1 << i_)) != 0;
        b[3] = (j & (1 << (i_ + 1))) != 0;
        if ((b[0] == b[1]) && (b[1] == b[2]) && (b[2] == b[3]))
            return false;
    }
    return true;
}

```

```

int main() {
    ifstream input; input.open("input.txt");
    ofstream output; output.open("output.txt");
    auto start = chrono::high_resolution_clock::now();
    double vm, rss;
    process_mem_usage(vm, rss);

    input >> n >> m;

    if (n > m) {
        int c = n;
        n = m;
        m = c;
    }

    int res = 0;
    int len = 1 << n;
    for (int i = 0; i < len; i++) {
        for (int j = 0; j < len; j++)
            dp[i][j] = calc(i, j);
    }

    for (int i = 0; i < len; i++)
        a[0][i] = 1;
    for (int i = 1; i < m; i++) {
        for (int j = 0; j < len; j++) {
            for (int k = 0; k < len; k++)
                a[i][j] += a[i-1][k] * dp[k][j];
        }
    }
    for (int i = 0; i < len; i++)
        res += a[m-1][i];
    output << res;

    auto end = chrono::high_resolution_clock::now();

    double diff = chrono::duration<double, std::milli>(end-start).count();
    cout << "Время выполнения: " << diff << endl;
    cout << "Использовано памяти: " << vm;

    return 0;
}

```

Для решения данной задачи используется динамическое программирование. Прием, использованный в данной задаче называется динамическое программирование на маске, маску используют, чтобы сэкономить и время и память.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```
2 1
RMAL input.tx
4
```

```
3 3
RMAL input.
out.txt" 1L,
22
```

Результат работы кода на максимальных и минимальных значениях:  
(скрины input output файлов)

```
1 1
RMAL input.tx
out.txt" 1L, 4
2
```

```
5 5
RMAL input.tx
out.txt" 1L, 4
19310334
```

18948899	04.03.2023 17:35:35	Филиппов Артём Эдуардович	0083	C++	Accepted	0,03	436 Kб
----------	---------------------	---------------------------	------	-----	----------	------	--------

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из	0.054034	5848



текста задачи		
Пример из задачи	0.053294	5848
Пример из задачи	0.058804	5848
Верхняя граница диапазона значений входных данных из текста задачи	0.126876	5848

Вывод по задаче: С битовыми операциями намного проще работать в C++ чем в питоне. Поэтому для некоторых задач резонно использовать C++.

## **Вывод**

Алгоритмы, основанные на динамическом программировании, жадном подходе используются в биоинформатике. Кто знает, может быть благодаря эффективным алгоритмам ученым удалось на 6 месяцев раньше открыть новую последовательность в ДНК, что поспособствовало созданию новых лекарств, которые могли спасти кому-то жизнь. В рядовых задачах программисту наврядли потребуются такие комплексные алгоритмы, но настоящему профессионалу не обойтись без них.