|  |  |
| --- | --- |
| **2.**  **from** itertools **import** \* **def** f(x,y,z,w):  **return** ((z <= x) == w) **or** y  **for** a1,a2,a3,a4 **in** product([0,1], repeat = 4):  table = [(1,0,0,a1),(1,a2,a3,1),(a4,0,1,1)]  **if** len(table) == len(set(table)):  **for** p **in** permutations(**'xyzw'**, r = 4):  **if** [f(\*\*dict(zip(p,t))) **for** t **in** table]== [1,0,0]:  print(p) | **3.**  ВПР(искомое\_значение, $диапазон, №\_столбца, 0) |
| **5. Условное изменение битовой записи**  *# если ищем max(N) - range(1000, 1, -1)* **for** n **in** range(1, 1000):  *# двоичное представление n (не забыть срезать [2:])* b = bin(n)[2:]   *# меняем разряды, если сумма цифр чётна (x%2 == 0)* **if** b.count(**'1'**) % 2 == 0:  b = b + **'0'** b = **'10'** + b[2:]  *# меняем разряды по условию, если сумма цифр нечётна* **else**:  b = b + **'1'** b = **'11'** + b[2:]   *# возвращаем R в десятичную CC (не забыть ,2)* r = int(b, 2)  *# ищем первое R, которое не меньше 16 (проверить знак)* **if** (r >= 16):  print(n) *# (убедиться возвращаем n или r)   # выходим из цикла* **break** | **5. Инверсия битовой записи**  *# если ищем max(N) - range(1000, 1, -1)* **for** n **in** range(1,256):  *# двоичное представление n (не забыть срезать [2:])* b = bin(n)[2:]  *# догоняем длину строки до 8 символов нулями слева* b = **'0'** \* (8 - len(b) ) + b  *#b = b.zfill(8)    # инвертируем разряды числа* b = b.replace(**'1'**,**'\*'**)  b = b.replace(**'0'**,**'1'**)  b = b.replace(**'\*'**,**'0'**)  *# если нужно перевернуть число* b = b[::-1]  *# возвращаем R в десятичную CC (не забыть ,2)*  r = int(b, 2)   *# ищем N при результате работы алгоритма R = 221* **if** r == 221:  print(n)  *# выходим из цикла* **break** |
| **6. Базовая черепаха**  **from** turtle **import** \* tracer(0) *# отключаем анимацию* hideturtle() *# убираем значёк черепахи* r = 50 *# задаём коэффициент масштабирования # подгоняем размер экрана под размер фигуры* screensize(50 \* r, 50 \* r)  *# задаём цвета границы (чёрный) и заливки (красный)* color(**'black'**, **'red'**) left(90) *# стартовый поворот под Кумир* begin\_fill() **for** i **in** range(8):  right(15)  forward(4 \* r)  right(30) end\_fill()  up() *# поднимаем перо, чтобы рисовались отдельные точки* **for** x **in** range(-50, 50):  **for** y **in** range(-50, 50):  goto(x\*r, y\*r) *# (не забыть умножить на r)* dot(5, **'blue'**) *# (уменьшить 5, если непонятная точка)* update() *# обновляем холст* done() *# завершаем работу (чтоб окно не закрылось)*  **7. Изображение и звук**  **from** math **import** log, ceil  *# Изображение* v = i \* a \* b *# a - длина изображения # b - ширина изображения # i - битовая глубина* i = ceil(log2(количество\_цветов))  *# Звук* v = k \* f \* i \* t *# k - количество каналов (моно=1, стерео=2) # f - частота дискретизации (1 КГц = 1 000 Гц) # i - битовая глубина* i = ceil(log2(количество\_уровней\_сигнала)) *# t - время (в секундах)* | **6. Автоматическая черепаха (пересечение без границ)**  **from** turtle **import** \* tracer(0) *# отключаем анимацию* hideturtle() *# убираем значёк черепахи* r = 20 *# коэффициент масштабирования* e = 0.1 *# размер области поиска пересечений (важно)* screensize(50 \* r, 50 \* r) *# размер экрана  # функция определения цвета области* **def** belong(item, color):  **return** any(i > 3 **and** canvas.itemcget(i, **'fill'**) == color **for** i **in** item)  left(90) *# стартовый поворот под кумир* color(**'black'**, **'red'**) *# цвета границы и заливки* begin\_fill() **for** i **in** range():  *# алгоритм отрисовки 1 фигуры по условию* end\_fill()  up() *# смещение между фигурами по условию* down()  color(**'black'**, **'green'**) begin\_fill() **for** i **in** range():  *# алгоритм отрисовки 2 фигуры по условию* end\_fill()  up() *# чтобы рисовались точки, а не линии* cnt = 0 *# счётчик точек* canvas = getcanvas() *# (не забыть создать канвас)* **for** x **in** range(-50\*r, 50\*r, r): *# теперь r внутри циклов* **for** y **in** range(-50\*r, 50\*r, r):  *# поиск всех пересечений в текущей точке* item = canvas.find\_overlapping(x-e, y-e, x+e, y+e)  *# если текущая точка и в первой, и во второй фигуре* **if** belong(item, **'red'**) **and** belong(item, **'green'**):  *# но не на их границе* **if not** belong(item, **'black'**):  cnt += 1 *# увеличиваем счётчик* goto(x,-y) *# (не забыть -y)* dot(5, **'blue'**)print(cnt) *# ответ* update() done() |
| **8. Комбинаторика**  cnt = 0 *#счётчик на старте всегда равен нулю* index = 1 *#значение индекса зависит от условия(обычно 1)*  *# Например, необходимо перебрать все четырёхзначные слова в шестеричной системе счисления* **for** b1 **in '12345'**: *# не забыть убрать 0* **for** b2 **in '012345'**: *# на остальные цифры условий нет* **for** b3 **in '012345'**: *# - алфавит шестеричной СС*  **for** b4 **in '012345'**:  **if** условие:  *# увеличиваем индекс или счётчик* cnt += 1  index += 1 | *номер первого слова в списке, начинающегося на У, в котором две буквы А НЕ стоят рядом из алфавита ПАРУС*  ok = **True** *# заглушка* index = 1 *# 5-буквенные слова - 5 циклов* **for** b1 **in 'АПРСУ'**: *# упорядочиваем в алфавитном порядке* **for** b2 **in 'АПРСУ'**:  **for** b3 **in 'АПРСУ'**:  **for** b4 **in 'АПРСУ'**:  **for** b5 **in 'АПРСУ'**:  *# собираем слово на каждой итерации* word = b1+b2+b3+b4+b5  *# если слово начинается на У и не содержит АА* **if** ((word[0]== **'У'**) **and** ( **not 'АА' in** word )):  *# печатаем его индекс* **if** ok:  print(index)  ok = **False** *# увеличиваем индекс на каждой итерации* index += 1 |
| **9. Электронные таблицы**  *# инициализируем счётчик* cnt = 0 *# построчно считываем файл* **for** s **in** open(**'9-24.csv'**):row = [int(item) **for** item **in** s.split(**';'**)]   *# пропускаем строки, где не ровно 1 повторяющийся 2 раза элемент* **if** len(row) != len(set(row)) + 1:  **continue** *# повторяющееся число* dup = sum(row) - sum(set(row))  *# сумма неповторяющихся чисел* sum\_unique = sum(row) - 2 \* dup   *# если сумма квадратов повторяющихся < квадрата суммы неповторяющихся* **if** dup\*\*2 + dup\*\*2 < sum\_unique\*\*2:  *# увеличиваем счётчик* cnt += 1 print(cnt) | *# 1. наибольшее из четырёх чисел меньше суммы трёх других; # 2. четыре числа можно разбить на две пары чисел с равными суммами.*  *# инициализируем счётчик* cnt = 0  *# построчно считываем файл* **for** s **in** open(**'9-24.csv'**):  *# формируем список из строки* row = [int(item) **for** item **in** s.split(**';'**)]   *# сортируем строку* row = sorted(row)   *# отбрасываем строки, которые невозможно разбить на пары чисел с равными суммами* **if** row[1] + row[2] != row[0] + row[3]:  **continue** *# если максимальное число меньше суммы остальных* **if** max(row) < sum(row) - max(row):  *# увеличиваем счётчик* cnt += 1 print(cnt) |
| **12. Строки**  *# Cумма числовых значений цифр строки, начинающейся с символа ">", # а затем содержащей 20 цифр 1, 10 двоек и 10 троек, расположенных в произвольном порядке.*  *# формуируем исходную строку (заметим, что результат не зависит от порядка цифр)* s = **'>'** + 20 \* **'1'** + 10 \* **'2'** + 10 \* **'3'** *# переписываем заданный алгоритм* **while '>1' in** s **or '>2' in** s **or '>3' in** s:  **if '>1' in** s:  s = s.replace(**'>1'**,**'21>3'**,1) *# (не забыть ,1)* **if '>2' in** s:  s = s.replace(**'>2'**,**'13>'**,1)  **if '>3' in** s:  s = s.replace(**'>3'**,**'1>2'**,1) *# избавляемся от нечислового символа '>' в строке* s = s[:-1] *# вычисляем сумму цифр строки с помощью генератора списка и функции sum* print(sum([int(x) **for** x **in** s])) | *# Количество значений n, при которых длина получившейся строки будет являться простым числом.*  *# функция проверки простоты двузначного числа* **def** is\_simple(x):  **for** d **in** range(2, x):  *# если проверяемое число x делится без остатка хотя бы на 1 число, большее единицы* **if** x % d == 0:  **return False** *# это число не простое  # если же число x не делится ни на одно число в диапазоне [2, x-1] - оно простое* **return True** *# создаём множество для хранения вариантов количества единиц в исходной строке* answers = set() *# перебираем все двузначные значения n* **for** n **in** range(10,100):  *# формируем исходную строку на каждой итерации цикла* s = n\***'1'** + 5\***'2'** + 8\***'3'** *# переписываем заданный алгоритм* **while …**:  **if …**:  s = s.replace(**'…'**, **'…'**, 1)  *# если длина получившейся строки - простое число* **if** is\_simple(len(s)):  *# добавляем число n в множество ответов* answers.add(len(s)) *# количество единиц в исходной строке* print(len(answers)) |
| **14. Системы счисления**  *# Количество значащих нулей в 64-ричной записи числа*  *вычисляем значение выражения в десятичной СС* n = 5 \* 512\*\*4000 + 4 \* 256\*\*3000 - 3\*64\*\*2000s = []*# список для хранения цифр 64-ричной записи* **while** n > 0:  *# каждый следующий разряд в начало представления* s = [n % 64] + s  n = n // 64 *# избавляемся от добавленного разряда* print(s.count(0)) *# подсчитываем количество нулей*  **15. Логические выражения**    *# логическое выражение* **def** f(x, a):  **return not**((x % a == 0) <= ((x % 44 != 0) <= (x % 8 != 0))) **and** (x % 8 == 0)  **for** a **in** range(1, 1000): *# если нужно наибольшее a - range(1000, 1,-1)  # если для всех x логическое выражение ложно* **if** all(f(x,a) == 0 **for** x **in** range(1,5000)):  print(a) *# печатаем ответ* **break** *# выходим из цикла* | *# перебираем все возможные СС, которыми может быть x* **for** x **in** range(21,100): *# x > 20, т.к. есть цифра K*  *# если при текущем x левая и правая части равны* **if** (5+2\*x) \* (7+x) == 3 + 20\*x + 2\*x\*\*2:  *# выводим на экран найденное значение x* print(x)  **break**    *# логическое выражение* **def** f(x,y,a):  **return** (y<a) <= ((y > 64-x) <= (y+24 <= x))  **for** a **in** range(100,1,-1):  *# если для всех x логическое выражение истинно* **if** all(f(x,y,a) == 1 **for** x **in** range(1,1000) **for** y **in** range(1,1000)):  print(a) *# печатаем ответ* **break** *# выходим из цикла* |
| P = {1, 3, 5, 8, 12, 14, 19, 24, 32, 44} Q = {2, 4, 7, 8, 12, 15, 21, 24, 27, 36}  *# ищем наименьшее возможное количество элементов, поэтому создадим пустое множество A # и будем добавлять в него недостающие для тождественной ложности значения* A = set()  **def** f(x,A):  **return** (x **not in** A) **and** ((x **not in** P) <= (x **in** Q))  **for** x **in** range(0,100):  **if** f(x,A): *# если при текущем значении x в A нарушается ложность выражения* A.add(x) *# добавляем недостающий x в множество A* print(len(A)) | **from** itertools **import** combinations **def** f(x, a1, a2):P = 27 <= x <= 80  Q = 20 <= x <= 32A = a1 <= x <= a2  **return** P <= (Q **or** (P <= A)) Ox = [x/4 **for** x **in** range(10\*4, 90\*4+1)]  *# список длин интервалов A* answers = []  *# перебираем всевозможные варианты начала и конца A* **for** a1, a2 **in** combinations(Ox, 2):  **for** x **in** Ox:  *# если при текущем значении x f(x,a1,a2) ложна* **if** f(x,a1,a2) == **False**:  **break** *# интервал нам не подходит**# если же при всех значениях x f(x,a1,a2) истинна* **else**: *# (else для for, не для if)  # добавляем длину интервала в список (ОКРУГЛИТЬ)* answers.append(round(a2) - round(a1))  print(min(answers)) |
| **16. Рекурсии с кэшированием**    **from** functools **import** lru\_cache @lru\_cache(maxsize=**None**) *# не забыть maxsize* **def** F(n):  **if** n <= 3:  **return** 1  **if** n > 3:  **return** F(n-3) + F(n-2) print(F(100)) | **16. Рекурсии с глубиной**    **import** sys sys.setrecursionlimit(2000)  **def** F(n):  **if** n == 0:  **return** 1  **return** n \* F(n-1)  *# сумма цифр целого числа* sum(map(int,str((F(1050))))) |
| **17. Числовые последовательности**  *# создаём поток для чтения файла 'имя\_файла.txt* f = open(**'имя\_файла.txt'**) *# считываем числовую последовательность в список* nums = [int(x) **for** x **in** f]  *# находим минимальный кратный n элемент* min19 = min([x **for** x **in** nums **if**  abs(x) % 19 == 0]) *# находим максимальный элемент последовательности, являющийся (полным) квадратом целого числа* maxSq = max([x **for** x **in** nums **if** x > 0 **and** (x\*\*0.5).is\_integer()])  *# получаем список модулей разности пар элементов, сумма которых меньше минимального, кратного 19* subs = [abs(a - b) **for** a, b **in** zip(nums, nums[1:]) **if** a + b < min19]  *# получаем список квадратов сумм троек элементов, cумма которых больше максимального полного квадрата* sums = [(a + b + c)\*\*2 **for** a, b, c **in** zip(nums, nums[1:], nums[2:]) **if** a + b + c > maxSq]  **19-21. Теория игр. 2 Кучи камней**  **from** functools **import** lru\_cache @lru\_cache(maxsize=**None**) **def** f(a,b):  *# условие завершения игры* **if** a + b >= 234:  **return** 0moves = [f(a+2,b), f(a,b+2), f(a\*3, b), f(a,b\*3)]loses = [x **for** x **in** moves **if** x <= 0]**if** loses: *# позиция - выигрышная за -max(loses) + 1* **return** 1 - max(loses)**else**: *# позиция - проигрышная за -max(moves) ходов* **return** - max(moves)  *# Вопрос 1 # начинаем поиск с конца, т.к. требуется найти максимальное значение s* **for** s **in** range(199, 1, -1):  *# Петя создал для Вани выигрышную в 1 ход позицию* **if** f(34+2,s) == 1 **or** f(34,s+2) == 1 **or** f(34\*3,s) == 1 **or** f(34,s\*3) == 1:  print(s)**break** *# Вопрос 2* wins = [] **for** s **in** range(1,200):  *# f(34, s) = 2 - позиции, в которых Петя выигрывает своим вторым ходом* **if** f(34,s) == 2:  wins +=[s] *# упорядочиваем найденные значения s по убыванию* print(max(wins), min(wins))  *# Вопрос 3* **for** s **in** range(1,200):  *# f(34, s) = -2 - позиции, в которых Ваня выигрывает своим вторым ходом* **if** f(34,s) == -2:  print(s) | *# создаём поток для чтения файла* f = open(**'имя\_файла.txt'**) *# считываем числовую последовательность в список*  nums = [int(x) **for** x **in** f]  *# находим минимальный элемент последовательности кратный 17 (минимум)* min17 = min([x **for** x **in** nums **if abs(**x) % 17 == 0])  *# получаем список модулей разности элементов, ровно один из которых нацело делится на минимум* subs = [abs(a - b) **for** a, b **in** zip(nums, nums[1:]) **if (**abs(a) % min17 == 0) + (abs(b) % min17 == 0) == 1]  *# печатаем количество элементов в списке, и его минимальный элемент* print(len(subs), min(subs))  **19-21. Теория игр. Запрет повторения хода**  **from** functools **import** lru\_cache @lru\_cache(maxsize=**None**) *# функция принимает дополнительный параметр type - id хода, сделанного на предыдущем шаге* **def** f(type, s):**if** s > 143:  **return** 0moves = []  *# если текущий ход первый в игре* **if** type == **'0'**:  *# допустимы все возможные действия* moves = [f(**'1'**, s+1), f(**'2'**, s\*2), f(**'3'**, s+3)]  *# если на предыдущем ходе в кучу добавили 1 камень* **if** type == **'1'**:  *# допустимы все действия, кроме добавления 1 камня* moves += [f(**'2'**, s\*2), f(**'3'**, s+3)]**if** type == **'3'**:moves += [f(**'1'**, s+1), f(**'2'**, s\*2)]**if** type == **'2'**:moves += [f(**'1'**, s+1), f(**'3'**, s+3)]  loses = [x **for** x **in** moves **if** x <= 0]  **if** loses:**return** 1 - max(loses)**else**:  **return** - max(moves)  *# Вопрос 1* **for** s **in** range(1,144):  *# f('0', s) = -1 - позиции, в которых Ваня выигрывает своим первым ходом* **if** f(**'0'**, s) == -1:  print(s)  *# Вопрос 2* wins = [] **for** s **in** range(1,144):**if** f(**'0'**, s) == 2:wins += [s]print(min(wins), max(wins)) |
| **23. Количество программ проходящих и не проходящих**  *# рекурсивная функция из 1 в 63 через 25, но не через 6*  **def** f(cur, fin):  *# исключаем траектории, не попадающие в пункт назначения или проходящие через 6* **if** cur > fin **or** cur == 6: **return** 0  *# если текущее число равно числу назначения - одна из траекторий найдена (т.к. мы попали в fin)* **if** cur == fin: **return** 1  *# если текущее число меньше числа назначения - продолжаем рекурсивный подсчёт суммы путей* **return** f(cur+2, fin) + f(cur\*3, fin)  *# строим мост через 25* print( f(1, 25) \* f(25, 63) ) | **23. Количество программ за заданное число команд**  *# рекурсивная функция из 5 в 44 за <=15 шагов # step - номер текущей команды в программе* **def** f(cur, fin, step):  *# если число fin получено не более чем за 15 команд - одна из программ найдена* **if** step <= 15 **and** cur == fin: **return** 1  *# если за 15 команд не удалось получить fin - обрыв* **if** step >= 15: **return** 0  *# если выполнено менее пятнадцати команд - продолжаем рекурсивный обход* **if** cur % 2 != 0:  *# для нечётных чисел доступны операции сложения и вычитания* **return** f(cur+1,fin,step+1) + f(cur-3,fin,step+1)  **if** cur % 2 == 0:  *# для чётных чисел доступны операции умножения и вычитания* **return** f(cur\*2,fin,step+1) + f(cur-3,fin,step+1) print( f(5,44,0) ) |
| **24. Анализ строк**  *# считываем текстовый файл в строковую переменную* s = open(**'24.txt'**).readline()  *# т.к. необходимо найти максимальную подстроку, не содержащую сочетания GO # нужно получить список подстрок исходной строки, полученных методом разбиения # каждого найденного сочетания GO по середине (между символами G и O). # А затем найти максимальную длину подстроки в этом списке.* **while 'GO' in** s:  *# меняем каждое сочетания 'GO' в строке на 'G O'* s = s.replace(**'GO'**, **'G O'**) *# находим максимальную из длин полученных подстрок* max(map(len,s.split()))  *# максимальное количество идущих подряд троек символов вида «буква + буква + цифра» (например: CA2)*  s = open(**'24.txt'**).readline()  *# максимальное количество идущих подряд троек вида: буква-буква-цифра* mx = 0  *# т.к. рассматриваются тройки - существует 3 варианта разбиения строки на тройки, # начиная с нулевого, первого и второго элемента. Рассмотрим все 3 варианта* **for** shift **in** range (3):  *# количество идущих подряд троек ББЦ в подстроке* k = 0  *# обходим строку по тройкам символов, начиная с заданного отступа shift* **for** i **in** range(shift,len(s)-2, 3):  *# если текущая тройка символов: буква-буква-цифра* **if** (s[i] **in 'ABC'**) **and** (s[i+1] **in 'ABC'**) **and** (s[i+2] **in '123'**):  *# увеличиваем количество идущих подряд троек на 1* k += 1  *# обновляем максимум* mx = max(k, mx)  **else**:  *# сбрасываем количество идущих подряд троек до 0* k = 0  *# печатаем максимальную длину искомой последовательности* print(mx) | *# минимальное число идущих подряд символов, среди которых комбинация END повторяется не менее пяти раз.* s = open(**'24.txt'**).readline()  *# получаем список подстрок, расположенных между двумя вхождениями комбинации END* lst = s.split(**'END'**)  *# отметим, что первый и последний элементы можно удалить, т.к. искомая строка должна иметь вид: # END - substr1 - END - substr2 - END - substr3 - END - substr4 - END, чтобы быть кратчайшей # А первая(последняя) подстрока не начинается(заканчивается) с комбинации END.* lst = lst[1:-1] lst = [**'END'**+a+**'END'**+b+**'END'**+c+**'END'**+d+**'END' for** a, b, c, d **in** zip(lst, lst[1:], lst[2:], lst[3:])]  *# находим минимальную из длин таких подстрок* print(min(map(len,lst)))  *# Максимальное количество идущих подряд символов, среди которых комбинация символов NM встречается хотя бы 10 раз.* s = open(**'24.txt'**).readline()  *# получаем список подстрок, разделённых комбинациями NM* s = s.split(**'NM'**) *# максимальная длина подстроки с десятью вхождениями NM* mx = -float(**'inf'**)  *# т.к. рассматриваются подстроки с 10 вхождениями NM - существует 10 вариантов # разбиения строки на такие подстроки. Рассмотрим все варианты* **for** shift **in** range(len(s)-10):  *# длина текущей подстроки с десятью вхождениями NM  # 20 - длина всех вхождений пар NM в строке, 2 - граничные M и N* k = 20 + 2  *# для каждого варианта разбиения длина текущей подстроки увеличивается* **for** i **in** range(shift, shift+11):  *# на длину 11 подстрок, расположенных вокруг символов NM* k += len(s[i])  *# обновляем максимальную длину подстроки, если текущая длина превышает максимум* mx = max(k,mx)  *# печатаем максимальную длину искомой последовательности* print(mx) |
| **25. Нетривиальные делители и маска**  *# Поиск всех чисел, не превышающих 10^6, соответствующих маске \*81\*,  # имеющих ровно 42 нетривиальных натуральных делителя.* **from** fnmatch **import** fnmatch *# функция для подсчёта делителей числа n* **def** D(n):  *# наибольший нетривиальный делитель числа* mx = 0  *# количество делителей* cnt = 0  *# обходим все числа до следующего целого после корня из числа n* **for** k **in** range(2,int(n\*\*0.5)+1):  *# если текущее число k - делитель числа n* **if** n % k == 0:  *# увеличиваем счётчик делителей на 1 (найден делитель k)* cnt += 1  *# если k - не корень из n, значит у k есть парный делитель n//k* **if** k != n//k:  *# увеличиваем счётчик делителей на 1 (найден делитель n//k)* cnt += 1  *# при первом нахождении делителя (найдены наименьший и наибольший делители)* **if** mx == 0:  *# записываем в mx наибольший нетривиальный делитель числа n* mx = n // k  *# возвращаем количество делителей и максимальный нетривиальный делитель* **return** cnt, mx  *# обходим все числа от 1 000 000 в порядке убывания* **for** n **in** range(10\*\*6, 0, -1):  *# если число соответствует заданной маске* **if** fnmatch(str(n), **'\*81\*'**):  *# находим количество делителей и максимальный нетривиальный делитель числа* cnt, mx = D(n)  *# если у числа ровно 42 нетривиальных делителя* **if** cnt == 42:  *# печатаем это число и его максимальный нетривиальный делитель* print(n, mx)  **26. Ячейки**  f = open(**'26.txt'**) k = int(f.readline()) n = int(f.readline()) *#k, n = map(int,f.readline().split())   # список багажей [время попытки сдачи, время возвращения]* bags = [] *# считываем данные по заявкам на сдачу багажей* **for** i **in** range(n):  bags += [list(map(int,f.readline().split()))] *# упорядочиваем время сдачи в хронологическом порядке* bags = sorted(bags) *# на случай сохранения очерёдности в файле #bags = sorted(bags, key = lambda x: x[0])  # список камер хранения (в каждой ячейке хранится время её очередного освобождения)* cells = [0]\*k  *# строго говоря, освобождать ячейки для поиска свободной ячейки не обязательно  # функция сдачи багажа в камеру хранения # в первую по индексу ячейку, время хранения багажа в которой меньше времени сдачи текущего багажа # записывается время завершения хранения текущего багажа* **def** put(bag):  **for** i **in** range(k): *# k = len(cells)  # если время хранения багажа в ячейке к текущей минуте уже истекло* **if** cells[i] < bag[0]:  *# записываем в эту ячейку время завершения хранения текущего багажа* cells[i] = bag[1]  *# возвращаем индекс ячейки, в которую удалось сдать багаж (начиная с нуля)* **return** i  *# если ни одной свободной ячейки не нашлось* **return** -1   *# суммарное количество принятых на хранение багажей* cnt = 0 *# индекс ячейки, в которую положили последний багаж* last\_index = 0  *# обходим список багажей* **for** bag **in** bags:  *# пробуем сдать багаж* index = put(bag)  *# если багаж принят на хранение* **if** index != -1:  *# увеличиваем счётчик сданных багажей* cnt += 1  *# сохраняем индекс ячейки, в которую удалось положить текущий багаж* last\_index = index *# печатаем ответ* print(cnt, last\_index+1) *# т.к. индексы ячеек в условии начинаются с 1* | **25. Сумма всех натуральных делителей**  *# Пусть S — сумма всех натуральных делителей целого числа, не считая самого числа и единицы. # программа, находит 4 наименьших числа, больших 5 500 000, для которых значение S кратно 123.  # функция, возвращающая сумму нетривиальных делителей числа n* **def** S(n):  *# множество делителей числа n* divs = set()  *# у каждого делителя числа n, меньшего, чем корень из n есть парный делитель, больший корня из n  # будем обходить все числа до корня, и при нахождении делителя k, добавлять его парный делитель n//k  # обходим все числа до следующего целого после корня из числа n* **for** k **in** range(2,int(n\*\*0.5)+1):  *# если текущее число k - делитель числа n* **if** n % k == 0:  *# добавляем число k d список делителей числа n* divs.add(k)  *# добавляем обратный ему делитель n//k в список делителей  # если k - корень из n (k = n//k) число уже есть в множестве и добавления не произойдёт* divs.add(n//k)  *# возвращаем сумму всех делителей числа n* **return**(sum(divs))  *# счётчик найденных чисел* cnt = 0 *# обходим все числа, большие чем 5 500 000* **for** n **in** range(5\_500\_000+1, 10\*\*10):  *# если сумма нетривиальных делителей числа кратна 123 и не равна нулю* **if** S(n) % 123 == 0 **and** S(n) != 0:  *# выводим на экран это число и результат деления суммы делителей этого числа на 123* print(n, S(n) // 123)  *# увеличиваем счётчик найденных чисел на 1* cnt += 1  *# если найдено уже 4 числа* **if** cnt == 4:  *# выходим из цикла* **break**  **26. Ячейки с очередями**  f = open(**'26.txt'**) n, m = map(int,f.readline().split()) *# список посетителей [приход, время обслуживания]* guests = []**for** i **in** range(m):  start, dur = map(int,f.readline().split())  guests.append([start, dur]) *# упорядочиваем время прихода в хронологическом порядке* guests = sorted(guests, key = **lambda** x: x[0])  *# список банкоматов (в каждом хранится время его очередного освобождения)* atms = [0]\*n *# список количеств клиентов, которых обслужил каждый банкомат* atms\_cnt = [0]\*n  *# функция назначения банкомату с соответствующим номером текущего клиента # в первый по индексу освободившийся банкомат записывается время завершения обслуживания текущего клиента* **def** assign(guest):  *# если в текущие сутки клиент не успеет подойти к банкомату:* **if** min(atms) > 1440:  *# возвращаем признак завершения* **return** -1  **for** i **in** range(n):  *# если время обслуживания клиента в текущем банкомате уже истекло* **if** atms[i] <= guest[0]:  *# записываем в этот банкомат время завершения обслуживания текущего клиента* atms[i] = max(atms[i], guest[0]) + guest[1]  *# увеличиваем число обслуженных клиентов этим банкоматом на 1* atms\_cnt[i] += 1  *# возвращаем номер банкомата, в котором будет обслужен текущий клиент* **return** i + 1    *# если ни одного свободного банкомата на момент прихода не нашлось  # находим индекс банкомата, который освободится первым* i = atms.index(min(atms))  *# записываем в этот банкомат время завершения обслуживания текущего клиента* atms[i] = max(atms[i], guest[0]) + guest[1]  *# увеличиваем число обслуженных клиентов этим банкоматом на 1* atms\_cnt[i] += 1  *# и возвращаем номер банкомата, в котором будет обслужен текущий клиент* **return** i + 1  *# обходим список клиентов* **for** guest **in** guests:  *# пробуем назначить клиенту банкомат* id = assign(guest)  *# если клиент не успел попасть к банкомату до завершения суток* **if** id == -1:  **break** *# суммарное количество обслуженных клиентов* min\_cnt = min(atms\_cnt) *# номер банкомата последнего клиента* last\_time = atms[atms\_cnt.index(min\_cnt)]  *# печатаем ответ* print(min\_cnt, last\_time) |