Муниципальный этап Всероссийской олимпиады школьников по информатике по профилю «Программирование»
2025-2026 учебный год

Разбор задач для 9-11 классов

Задача A «Ноутбуки» (100 баллов

Как видно из условия задачи, наибольшее число ноутбуков равно min(a, b).

Для нахождения наименьшего возможного числа подсчитаем количество ноутбуков, оснащенных восьмиядерным процессором и имеющих объем оперативной памяти 16 Гб. Пусть A и B — множества ноутбуков с восьмиядерным процессором и с объемом оперативной памяти 16 Гб соответственно, тогда $|A \cap B| = |A| + |B| - |A \cup B| = a + b - n$.
(Здесь |A| — количество элементов в множестве A.) Если $a + b - n < 0$, это число следует считать равным 0. Другими словами, наименьшее возможное число ноутбуков равно $\max(0, a + b - n)$.

Задача B «Галактические шифры»

Необходимо реализовать алгоритм шифрования, предложенный в условии задачи. Для этого необходимо вычислить количество одинаковых соседних элементов и заменить их последовательность в соответствии с правилами из условия.

Пример решения задачи на языке Python:

```
s = input()

k = 1

arr = []

for i in range(len(s) - 1):

    if s[i] == s[i + 1]:

        k += 1

    else:

        arr.append((s[i], k))

        k = 1

arr.append((s[-1], k))

for x in arr:

    if x[1] == 1:

        print(x[0], end='')

    else:

       print(x[0], x[1], sep='', end='')
```

Задача C «Сейф профессора Кванта»

Сразу обратим внимание на особый случай – p = 0, в этом случае представляем число максимальным количеством цифр 9, первой цифрой можем поставить остаток для получения суммы при всех остальных цифрах 9 в числе. При p != 0 необходимо его разложить на множители от 2 до 9 различными способами. Если не нашлось разложений числа на такие множители – то не существует числа с заданным произведением цифр. В случае, если разложение нашлось – будем проверять сумму множителей каждого из разложений:
- если сумма больше заданного s, то переходим к следующему разложению;
- если сумма равна s – найдены цифры одного из возможных чисел;
- если сумма множителей меньше s, то можем дополнить число слева единицами до заданной суммы.

Решение на языке Python:

```
def get_digits(p, k, d):
    if p == 1:
```

```
            res.append(sorted(d))
        else:
            for i in range(k, 1, -1):
                if p % i == 0:
                    get_digits(p // i, i, d + [i])


s, p = [int(x) for x in input().split()]
if p == 0:
    digits = []
    while s >= 10:
        digits.append(9)
        s -= 9
    digits.append(s)
    digits.sort()
    print(digits[0], 0, *digits[1:], sep='')
else:
    res = []
    get_digits(p, 9, [])
    res.sort(key=len)
    if len(res) == 0 and p != 1:
        print(-1)
    elif p == 1:
        print('1' * s)
    else:
        i = 0
        ans = []
        while i < len(res):
            if sum(res[i]) > s:
                pass
            elif sum(res[i]) == s:
                g = ''.join([str(w) for w in res[i]])
                ans.append(g)

            else:
                g = ''.join([str(w) for w in res[i]])
                ans.append('1' * (s - sum(res[i])) + g)
            i += 1
        if len(ans) == 0:
            print(-1)
        else:
            print(min(ans, key=lambda d: (len(d), int(d))))
```

Задача D «Подбор комплектующих»

Необходимо сортировать стоимости комплектующих, после этого методом двух указателей найдем искомую сумму:

```
n, x = map(int, input().split())
a = list(map(int, input().split()))
a.sort()
R = n - 1
best = 0
for L in range(n):
    while R > 0 and a[L] + a[R] > x:
        R -= 1
    if L != R and a[L] + a[R] <= x:
        best = max(a[L] + a[R], best)
print(best)
```

Задача E «Айсберг - 3»

Введём систему координат, в которой ось OX направлена вправо (на восток), ось OY — вверх (на север), а начало координат находится в центре квадрата (начальная позиция робота).

Чтобы набрать частичный балл, можно промоделировать весь путь робота по одному шагу. Если координаты робота (x, y), то он вышел за пределы периметра, когда |x| = K или |y| = K. Закодируем направления движения числами 0 (север), 1 (восток), 2 (юг), 3 (запад), тогда робот меняет направления движения в порядке 0, 1, 2, 3, 0, 1, 2, 3 и изменение направления движения осуществляется инструкцией dir = (dir + 1) % 4. В массивах DX и DY хранятся координаты векторов перемещения на один шаг в данном направлении, то есть если направление движения есть dir, то по оси OX робот сместится на значение DX[dir], а по оси OY — на значение DY[dir]. Робот будет перемещаться, пока не дойдёт до границ периметра, при этом общее число шагов подсчитывается в переменной steps, а число шагов, которое он совершил после последнего поворота, в переменной last_segment. Если значение last_segment стало равно числу шагов, которое робот совершает в данном направлении (оно хранится в count[dir]), то нужно сделать поворот — изменяется значение dir и обнуляется значение last_segment. Пример такого решения:

```
count = [int(input()) for i in range(4)]
k = int (input( ))
DX = [ 0 , 1 , 0 , −1]
DY = [ 1 , 0 , −1, 0 ]
x = 0
y = 0
steps = 0
last_segment = 0
dir = 0
while abs(x) != k and abs(y) != k:
    x += DX[dir]
    y += DY[dir]
    steps += 1
    last_segment += 1
    if last_segment == count[dir]:
        dir = (dir + 1) % 4
        last_segment = 0
print(steps)
```

Это решение можно улучшить, если моделировать перемещения не одиночными шагами, а сразу же на весь отрезок длины count[dir]. В этом случае к значению x будет добавляться DX[dir] * count[dir], к значению y будет добавляться DY[dir] * count[dir], и сразу после этого необходимо сделать поворот dir = (dir + 1) % 4. При таком моделировании, после очередного перемещения робот может выйти за границы квадрата, поэтому цикл продолжается, пока обе координаты по модулю меньше k, а после окончания цикла нужно вычесть излишне пройденные вне квадрата шаги (то есть если робот оказался в точке (x, y) и |x| > k, то нужно вычесть |x| −k, если же |y| > k, то нужно вычесть |y| − k излишне пройденных шагов).

Для решения на полный балл нужно просто перебрать четыре направления, в которых робот может выйти за предела периметра, и для каждого из этих направлений посчитать, когда это произойдёт. Пусть в результате перемещений на A шагов на север, B шагов на восток, C шагов на юг и D шагов на запад робот оказался в точке (x, y), где y = A − C, x = B − D. Назовём это «циклом». Тогда после второго цикла робот окажется в точке (2x, 2y). после третьего цикла — в точке (3x, 3y) и т.д. Заметим, что если y > 0, то робот может выйти за пределы периметра в направлении на север, а если y < 0 — то в направлении на юг. Если x > 0, то робот может выйти за пределы периметра в направлении на восток, а если x < 0 — то в направлении на запад. Разберём случай y > 0, то есть когда робот выйдет за пределы периметра в северном направлении. Посчитаем, сколько циклов пройдёт, прежде чем робот выйдет за пределы периметра в этом направлении. На каждом цикле робот перемещается на север на y шагов, а затем он делает A шагов на север, прежде чем начнёт возвращаться на юг. При этом координата робота должна оказаться не меньше чем K. Общее число шагов, которое сделает робот, будет равно n(A + B

+ C + D) + K − ny (за каждый из n циклов робот проходит по A + B + C + D шагов, после этого он оказывается в точке (nx, ny) и чтобы дойти до границы периметра ему понадобится K−ny шагов).
Рассмотрим случай x > 0. В этом случае робот перемещается на восток на B шагов, потом возвращается на запад на D шагов. Общее число шагов равно m(A + B + C + D) + A + K − mx. Похожим образом надо рассмотреть возможность выхода за пределы периметра в южном и западном направлениях и взять минимальное количество шагов, необходимое для выхода во всех четырёх направлениях. Также отметим, что на первом цикле робот может выйти в каждом из четырёх направлений. Например, если A > K, то робот сразу же выйдет за пределы периметра при первом движении на север, даже если потом окажется, что y < 0. Поэтому отдельно нужно рассмотреть возможность выхода за пределы периметра во всех четырёх направлениях на первом цикле.

```
count = [int(input()) for i in range(4)]
k = int (input())
DX = [ 0 , 1 , 0 , −1]
DY = [ 1 , 0 , −1, 0 ]
x = 0
y = 0
steps = 0
flag = True
for i in range(4):
    x += DX[i] ∗ count [i]
    y += DY[i] ∗ count [i]
    steps += count[i]
    if abs ( x ) >= k:
        print(s t e p s − abs ( x ) + k)
        flag = False
        break
    if abs(y) >= k:
        print (steps − abs(y) + k)
        flag = False
if flag:
    ans = 10 ∗∗ 19
    if y > 0:
        p = (k − count [ 0 ] + y − 1 ) // y
        ans = min(ans , steps ∗ p + k − p ∗ y )
    if x > 0:
        p = (k − count [ 1 ] + x − 1 ) // x
        ans = min(ans , steps ∗ p + k − p ∗ x + count[0])
    if y < 0:
        p = (k − count [ 2 ] + count [ 0 ] − y − 1 ) // −y
        ans = min(ans , steps ∗ p + k + p ∗ y + count[0] ∗ 2 + count[1])
    if x < 0 :
        p = (k − count [ 3 ] + count [ 1 ] − x − 1 ) // −x
        ans = min(ans , steps ∗ p + k + p ∗ x + count [ 0 ] + count [ 1 ] ∗ 2 + count [ 2 ] )
    print(ans)
```