I. Рекомендации по оцениванию

Решением задачи является программа, написанная на одном из доступных на олимпиаде языков программирования. Для проверки и оценивания решений жюри использует автоматическую тестирующую систему. На проверку отправляется исходный текст программы. При отправке решения на проверку участник указывает, с использованием какого языка программирования и компилятора выполнено решение. Разные решения, отправленные на проверку, могут использовать разные языки программирования и/или компиляторы.

Программа запускается на тестах. Для каждого теста, на котором был выполнен запуск, устанавливается результат выполнения на этом тесте. Верный ответ на тест, выданный при соблюдении указанных в условии задачи ограничений, соответствует результату OK.

Когда программа запускается, ей указанным в условии задачи способом передаются входные данные. Для ввода данных используется стандартный поток ввода.

В условии каждой задачи приведены примеры входных и выходных данных для этой задачи. Решение участника запускается на тестах из примеров, приведенных в условии задачи, результат работы на этих тестах сообщается участнику.

Каждая задача оценивается максимум в 100 баллов. Каждый пройденный тест (за исключением тестов из условия) оценивается в 5 баллов. Оценка за задачу вычисляется по формуле: (кол-во пройденных тестов)*5.

II. Краткие рекомендации по решению задач, примеры решений

В состав пакета для каждой задачи входят решения (в электронном виде) на языках программирования Pyton и Free Pascal.
Ниже представлены краткие рекомендации к решению задач. Также в состав комплекта входит авторский видеоразбор (ссылки на просмотр в тестирующей системе).

Задача 1. Соседи

Задача на целочисленное деление и проверку условий

Разберем частные случаи.

1) k = 1. Пусть h – неизвестная высота дома в этажах. Тогда дом будет выглядеть так:

h	2h	3h
...	...	...
3	h+3	2h+3
2	h+2	2h+2
1	h+1	2h+1 ...

Видно, что номера квартир на одном этаже в разных подъездах отличаются на h. Поэтому h = m – n.

А номер этажа будет равен остатку от деления номера квартиры (любой – n или m) на h (если он равен нулю, то ответ – h).

2) k = 2. Пусть h – неизвестная высота дома в этажах. Тогда дом будет выглядеть так:

2h-1	2h	4h-1	4h	6h-1	6h
...	...	...	...	...	...
5	6	2h+5	2h+6	4h+5	4h+6
3	4	2h+3	2h+4	4h+3	4h+4
1	2	2h+1	2h+2	4h+1	4h+2 ...

Видно, что номера квартир на одном этаже в разных подъездах отличаются на 2h-1. Поэтому 2h-1 = m – n, откуда h = (m-n+1)/2

"Перенесем" квартиру n в первый подъезд: найдем остаток p от деления n на 2h (если он равен 0, то p = 2h).

Искомый номер этажа ans = p / 2

3) Полное решение. В условии не указана высота дома, найдем её.

Пусть h - искомая высота, тогда n + kh + k - 1 = m

Тогда высота дома h = (m - n + k - 1) / k

"Перенесем" квартиру n в первый подъезд: найдем остаток p от деления n на kh (если он равен 0, то p = kh)

Искомый номер этажа ans = p / k

Пример программы:

```
k = int(input())
n = int(input())
m = int(input())
h = (m - n + k - 1) // k
p = n % (h * k)
if p == 0:
    p = h * k
ans = p // k
print(ans)
```

Задача 2. Сложность слова

Задача на строки. Задача на реализацию.

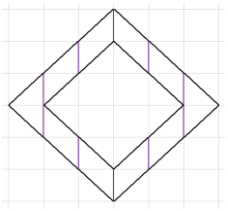
Выполним указанные действия: переберем все сочетания соседних букв и просуммируем их сложность. Результат умножим на длину слова.

Пример программы

```
p = int(input())
s = input()
vowels = 'aeiouy'
consonants = 'bcdfghjklmnpqrstvwxz'
score = 0
if len(s) == 1:
    score = 1
else:
    for j in range(1, len(s)):
        if s[j - 1] in consonants and s[j] in vowels:
            score += 1
        elif s[j - 1] in vowels and s[j] in consonants:
            score += 2
        elif s[j - 1] in vowels and s[j] in vowels:
            score += 5
        else:
            score += 7
ans = score * len(s)
print(ans)
```

Задача 3. Заштрихованный квадрат

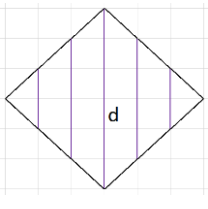
Задача на сокращение перебора. Задача на арифметическую прогрессию.



1) Разберем частный случай a – b = 2.

В квадрате проводятся a – 1 штриховых линий сверху и столько же снизу. Поэтому ответ на задачу 2a – 2.

2) Полное решение. Научимся сначала определять длину всех линий внутри одного квадрата с диагональю d. Очевидно, она будет равна 2 + 4 + 6 + … + d-2 + d + d-2 + … + 6 + 4 + 2. Для малых значений d эту сумму можно найти перебором, для больших – вывести формулу.



Если вынести двойку за скобки, сумма разбивается на две арифметические прогрессии с разностью 1 и первым членом 1, одна заканчивается числом d/2-1, вторая – d/2. Применим формулу суммы членов арифметической прогрессии: ans = 2 * (d/2 * (d/2 + 1) / 2 + d/2 * (d/2 – 1) / 2), преобразуем и сократим: d/2 * (d/2 – 1) + d/2 * (d/2 + 1) = d * d / 2

Тогда искомое значение равно a * a / 2 – b * b / 2 (или другие эквивалентные формулы).

Пример программы

```
a = int(input())
b = int(input())
ans = 2 * ((a // 2) ** 2 - (b // 2) ** 2)
print(ans)
```

Задача 4. Сумма степеней двоек

Задача на системы счисления. Задача на строки.

1) Разберем частный случай $n \leq 1000$.
Эту подзадачу можно решить полным перебором всех возможных вариантов первого и последнего слагаемых в сумме 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 + 512
Пример программы:

```
n = int(input())
ans = 0
for i in range(1, n + 1):
    for start in range(0, 10):
        for finish in range(start + 1, 11):
            summa = 0
            for j in range(start, finish):
                summa += 2 ** j
            if summa == i:
                ans += 1
print(ans)
```

2) Разберем частный случай $n \leq 100000$.
Заметим, что если число можно представить в виде $2^m + 2^{m-1} + 2^{m-2} + 2^{m-k}$, то в двоичном представлении числа n сначала идут только единицы, а потом – только нули. Переберем все числа от 1 до n и проверим, выполняется ли это условие.
Пример программы:

```
def to_bin(n):
    s = ''
    while n:
        s = str(n % 2) + s
        n //= 2
    for i in range(1, len(s)):
        if s[i - 1] == '0' and s[i] == '1':
            return False
    return True

n = int(input())
ans = 0
for i in range(1, n + 1):
    if to_bin(i):
        ans += 1
print(ans)
```

3) Полное решение.

Переведем данное нам число в двоичную систему счисления и переосмыслим условие: сколько существует двоичных чисел, не превосходящих данное, таких что в их записи сначала идут только единицы, а потом – только нули? Пусть длина полученного числа равна k. Очевидно, что все числа, у которых длина меньше длины полученного числа дадут нам следующее число решений:

Однозначные – 1 (1)

Двузначные – 2 (10 и 11)

Трехзначные – 3 (100, 110 и 111)
…
k-1 значные – k-1
Всего их будет k * (k-1) / 2 (опять формула суммы арифметической прогрессии).

Осталось подсчитать количество подходящих чисел длины k. Оно будет равно непрерывному количеству единиц с начала полученного числа.

Например, при n = 10. Переведем в двоичное представление:
$10 = 1010_2$, k = 4.
Однозначные – 1 (1)
Двузначные – 2 (10 и 11)
Трехзначные – 3 (100, 110 и 111)
Всего их 4 * (4-1) / 2 = 6. И есть еще одно (так как двоичное представление n начинается с одной единицы) подходящее число длины k: 1000. Всего 7 чисел.
Пример программы:

```
n = int(input())
s = bin(n)[2:]
ans = len(s) * (len(s) - 1) // 2
i = 0
while i < len(s) and s[i] == '1':
    ans += 1
    i += 1
print(ans)
```

Задача 5. Городки

Задача на массивы. Задача на поиск максимума. Задача на два указателя.

Разберем частные случаи.

1) n = 1. У нас есть единственная чурка. Если её длина четная – распилим её и получим две чурки половинной длины. Если нет – одна чурка исходной длины.
2) n = 2. У нас есть две чурки. Рассмотрим варианты:
    a. Чурки одной длины, длина четная. Пилим обе чурки, получаем 4 чурки половинной длины.
    b. Чурки одной длины, длина нечетная. В ответе 2 чурки исходной длины.
    c. Чурки разного размера, большая в два раза больше меньшей. Пилим большую пополам, получаем три чурки меньшего размера.
    d. Чурки разного размера, большая не в два раза больше меньшей. Меньшая чурка четной длины. Пилим меньшую пополам, получаем две чурки меньшей длины.
    e. Чурки разного размера, большая не в два раза больше меньшей. Меньшая чурка нечетной длины, большая – четной. Пилим большую пополам, получаем две чурки половины большей длины.
    f. Чурки разного размера, обе нечетной длины. Для игры возьмем одну чурку меньшей длины.
3) n, $a_i \le 10^3$. Пусть p – наибольшее значение среди чурок. Заведем массив длины 2p и заполним его по принципу: значение элемента массива равно количество чурок длины, равных индексу (половина массива останется равной 0, поскольку у нас нет

чурок длиннее, чем p, но это нужно для корректного дальнейшего прохода по массиву, чтобы не писать лишних условий).
Далее используем стандартный алгоритм поиска максимума. Пусть maximum – наибольшее количество, d – соответствующая длина. В начале работы их значения равны нулю. Пройдем по всем возможным значениям от 1 до p и попробуем сделать как можно больше цилиндров текущей длины i. Очевидно, что это значение будет равно значению массива с индексом i плюс удвоенному значению массива с индексом 2i. Если это значение превышает maximum – обновляем его и запоминаем текущее i в переменную d.
Пример программы:

```
n = int(input())
L = list(map(int, input().split()))
p = L[-1]
M = [0 for i in range(p * 2 + 1)]
for i in L:
    M[i] += 1
maximum = 0
d = 0
for i in range(1, p + 1):
    if M[i] + M[2 * i] * 2 > maximum:
        maximum = M[i] + M[2 * i] * 2
        d = i
print(maximum, d)
```

4) Полное решение. Исходный массив будет удобнее привести к виду пар длина-количество (получится один двумерный массив или два одномерных массива). Заведем переменные: maximum – наибольшее количество, d – соответствующая длина, label – второй указатель. После этого пройдем по получившемуся массиву.
Если текущее значение длины нечетное, значит распилить эти чурки нельзя. Обновим максимум, если текущее количество больше максимума.
Если текущее значение длины четное, есть два варианта.
Первый: не существует чурок с длиной в два раза меньше текущей (для проверки этого будем двигать второй указатель, пока длина чурок, на который он указывает, в два раза меньше текущей длины). Обновим максимум, если удвоенное текущее количество больше максимума.
Второй: такие чурки существуют. Обновим максимум, если удвоенное текущее количество плюс количество чурок в два раза меньшей длины больше максимума.
Во всех случаях при обновлении максимума будем запоминать соответствующую ему длину.

Пример программы:

```
n = int(input())
L = list(map(int, input().split()))
M = [[L[0], 1]]
for i in range(1, n):
    if L[i] == M[-1][0]:
        M[-1][1] += 1
    else:
        M.append([L[i], 1])
label = 0
maximum = 0
d = 0
for i in range(len(M)):
    if M[i][0] % 2 == 0:
        while M[label][0] * 2 < M[i][0]:
            label += 1
        if M[label][0] * 2 == M[i][0]:
            if M[label][1] + M[i][1] * 2 > maximum:
                maximum = M[label][1] + M[i][1] * 2
                d = M[label][0]
        else:
            if M[i][1] * 2 > maximum:
                maximum = M[i][1] * 2
                d = M[i][0] // 2
    else:
        if M[i][1] > maximum:
            maximum = M[i][1]
            d = M[i][0]
print(maximum, d)
```

III. Набор тестов к задачам

В состав пакета для каждой задачи входит тесты из условия и 20 уникальных тестов, на которых рекомендуется оценивать решение участников олимпиады, а также верные ответы на эти тесты.

В тестирующей системе также проверяются решения участников олимпиады на тестах из условия задачи. Баллы за такие тесты не даются.