Задача 1. Автобусные остановки

Остановки расположены в точках, кратных k, поэтому от последней пройденной остановки Света прошла $n\%k$ метров (здесь % — операция взятия остатка от деления, в языке Pascal это операция mod). А до следующей остановки нужно идти $k - n\%k$ метров. Ответом будет минимум из этих величин.

Пример решения.

```
k = int(input())
n = int(input())
print(min(n % k, k - n % k))
```

Задача 2. Наборы пирожных

В одной коробке лежит 3 пирожных, поэтому $a + b$ должно делится на 3, иначе ответ -1.

Если $a + b$ делится на 3, то посчитаем общее количество коробок $n = (a + b)/3$. Пусть n_1 — количество коробок первого вида, а n_2 — количество коробок второго вида. Посмотрим на коробки первого вида. В каждой коробке лежит минимум один эклер, то есть в n коробках лежит n эклеров. В коробке первого вида лежит один дополнительный эклер, поэтому количество коробок первого вида должно быть равно $a - n$. При этом $a \geqslant n$, иначе задача не имеет решения. Аналогично, количество коробок второго вида равно $b - n$. То есть если выполнены условия $a \geqslant n$ и $b \geqslant n$, то ответом будут числа $a - n$ и $b - n$.

Пример решения.

```
a = int(input())
b = int(input())

if (a + b) % 3 != 0:
    print(-1)
else:
    n = (a + b) // 3
    if a >= n and b >= n:
        print(a - n, b - n)
    else:
        print(-1)
```

Задача 3. Мирные ладьи

Решение, в котором создаётся двумерный массив размером $n \times n$ будет набирать 60 баллов. В этом массиве отмечаются клетки, занятые ладьями по строкам (для каждой строки считывается номер столбца, где стоит ладья, и этот элемент отмечается специальным значением, например, числом 1). Затем массив обходится по столбцам, так как при повороте строки становятся столбцами, в каждом столбце находится значение, которое равно 1 и выводится ответ.

Для того, чтобы набрать полный балл, необходимо понять, что происходит с координатами ладьи при повороте. Если ладья стояла в клетке (i, j), то есть в i-й строке в j-м столбце, то после поворота она окажется в j-й строке и в столбце $n + 1 - i$. То есть если организовать цикл по номеру строки i от 1 до n и в этом цикле считать номер столбца j, то эта ладья перейдёт в клетку $(j, n + 1 - i)$, значит, ответом для j-й строки будет число $n + 1 - i$. Поскольку нам необходимо вывести ответ в порядке возрастания номеров строк ответа от 1 до n, то мы будем сохранять ответ для j-й строки в элементе массива $a[j] = n + 1 - i$, а потом выведем ответ.

Такое решение набирает 100 баллов.

```
n = int(input())
a = [0] * (n + 1)
for i in range(1, n + 1):
    j = int(input())
    a[j] = n + 1 - i
for i in range(1, n + 1):
    print(a[i])
```

Задача 4. Таблица

40 баллов можно набрать, если написать решение, полностью моделирующее заполнение этой таблицы. Пусть x — текущее число, которое будет меняться от 1 до n, i и j — номера клеток, в которое будет записываться это число. Далее нужно корректно обработать, как меняются значения i и j при увеличении x на 1. Здесь придётся разобрать 6 случаев: 4 случая при движении по стороне квадрата (чётные и нечётные квадраты обходятся в разных направлениях, у квадрата две стороны), и два случая — когда при обходе достигается верхняя или левая сторона таблицы и происходит переход к следующему квадрату.

Пример решения на языке C++. Сложность этого решения $O(n)$.

```
#include<iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;
    int i = 1;
    int j = 1;
    int di = -1;
    int dj = 0;
    int x = 1;
    while (x != n)
    {
        if (i == 1)
        {
            if (di == -1)
            {
                di = 0;
                dj = 1;
            }
            else
            {
                di = 1;
                dj = 0;
            }
        }
        else if (j == 1)
        {
            if (dj == -1)
            {
                dj = 0;
                di = 1;
            }
            else
            {
                di = 0;
                dj = 1;
            }
        }
        else if (i == j)
        {
            if (di == 1)
```

```
            {
                di = 0;
                dj = -1;
            }
            else
            {
                di = -1;
                dj = 0;
            }
        }
        i += di;
        j += dj;
        ++x;
    }
    cout << i << " " << j << endl;
}
```

Можно написать решение сложность $O(\sqrt{n})$, если посчитать, сколько полных квадратов будет заполнено в таблице. Пусть k — сторона полностью заполненого квадрата. Тогда $k^2 \leqslant n$, т.к. в этом квадрате будет k^2 чисел. Будем увеличивать значение k, пока не найдём такое максимальное k, что $k^2 < n$, это размер полностью заполненных квадратов (если не рассматривать саму клетку, где находится число n). Наше число n будет находиться в квадрате со стороной $k+1$. Дальше необходимо разобрать случаи — на какой стороне лежит это число и в каком направлении обходится квадрат и вывести результат.

Пример решения сложности $O(\sqrt{n})$ на языке Python (60 баллов).

```
n = int(input())
k = 1
while k * k < n:
    k += 1
k -= 1
if n - k ** 2 <= k + 1:
    x = k + 1
    y = n - k ** 2
else:
    x = (k + 1) ** 2 - n + 1
    y = k + 1
if k % 2 == 0:
    print(x, y)
else:
    print(y, x)
```

Это решение не набирает полный балл, так как если $n \approx 10^{18}$, то $\sqrt{n} \approx 10^9$, а такое число операций нельзя выполнить за 0,5 секунд.

Для решения на полный балл необходимо найти быстро корень из числа n, то есть такое наибольшее целое k, что $k^2 < n$. Хочется воспользоваться функцией извлечения квадратного корня и написать такое решение:

```
n = int(input())
k = int((n - 1) ** 0.5)
if n - k ** 2 <= k + 1:
    x = k + 1
    y = n - k ** 2
else:
    x = (k + 1) ** 2 - n + 1
    y = k + 1
if k % 2 == 0:
```

```
    print(x, y)
else:
    print(y, x)
```

Такое решение будет работать быстро, но не всегда правильно. Такое решение будет набирать 90 баллов. Дело в том, что вычисление корня требует работы с действительными числами, а стандартный тип данных для действительных чисел (тип float в Python, тип double в C++) — это так называемые числа "двойной точности", точность их представления порядка 15–16 десятичных знаков. А во входных данных число дано с точностью в 18 знаков, поэтому при извлечении корня из n число n будет преобразовано к действительному типу, поэтому после вычисления корня и переводу результата к типу int может быть получен неправильный ответ.

Необходимо каким-то образом научиться точно вычислять целочисленный квадратный корень из чисел порядка 10^{18}. Для этого есть несколько вариантов.

а) Извлечь корень при помощи действительных чисел, преобразовать к типу int, а затем немного "пошевелить" ответ, изменяя его на ± 1, чтобы устранить возможную ошибку работы с действительными числами. Пример такого решения (на 100 баллов).

```
n = int(input())
k = int((n - 1) ** 0.5)
if (k + 1) ** 2 <= n - 1:
    k += 1
if k ** 2 > n - 1:
    k -= 1
if n - k ** 2 <= k + 1:
    x = k + 1
    y = n - k ** 2
else:
    x = (k + 1) ** 2 - n + 1
    y = k + 1
if k % 2 == 0:
    print(x, y)
else:
    print(y, x)
```

б) Использовать двоичный поиск для извлечения корня, в этом случае совсем не используются действительные числа. Пример такого решения (на 100 баллов).

```
n = int(input())
l = 0
r = 10 ** 9 + 1
while r != l + 1:
    m = (l + r) // 2
    if m * m < n:
        l = m
    else:
        r = m
k = l
if n - k ** 2 <= k + 1:
    x = k + 1
    y = n - k ** 2
else:
    x = (k + 1) ** 2 - n + 1
    y = k + 1
if k % 2 == 0:
    print(x, y)
else:
    print(y, x)
```

в) Использовать действительные числа расширенной точности, что возможно не во всех языках программирования. Например, на языке C++ это тип данных long double (при использовании компилятора GNU C++), на языке Pascal это тип данных extended (при использовании компилятора Free Pascal).

Пример такого решения на языке C++.

```
#include<iostream>
#include<cmath>

using namespace std;

int main()
{
    long long n;
    cin >> n;
    long long k = sqrtl(n - 1);
    long long x, y;
    if (n - k * k <= k + 1)
    {
        x = k + 1;
        y = n - k * k;
    }
    else
    {
        x = (k + 1) * (k + 1)  - n + 1;
        y = k + 1;
    }
    if (k % 2 == 0)
        cout << x << " " << y << endl;
    else
        cout << y << " " << x << endl;
}
```

Задача 5. Agar.io

Сначала несколько общих соображений. Если мы хотим понять, может ли победить какая-то выбранная бактерия, то эта бактерия будет поедать другие бактерии, а другие бактерии поедать друг друга не будут (лучше расправиться с другими бактериями, если их размеры не будут увечиваться). Кроме того, все остальные бактерии лучше поедать в порядке возрастания их размеров (будем есть сначала маленькие бактерии, это позволит нарастить размер и съесть бактерии побольше).

Решение на 60 баллов — для каждой бактерии будем моделировать процесс. Будем перебирать все оставшиеся бактерии в порядке возрастания их размеров, и проверять, сможет ли наша бактерия съесть следующую в этом списке, если она уже съела все предыдущие. Такое решение будет иметь сложность $O(n^2)$. Пример такого решения на языке C++.

```
#include<iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;
    if (n == 1)
    {
        cout << 1 << endl;
```

```
        return 0;
    }
    int a[n];
    for (int i = 0; i < n; ++i)
        cin >> a[i];
    for (int i = 0; i < n ; ++i)
    {
        long long s = a[i];
        int j;
        for (j = 0; j < n; ++j)
        {
            if (j == i)
                continue;
            if (s > a[j])
                s += a[j];
            else
                break;
        }
        cout << (j == n) << "\n";
    }
}
```

Для того, чтобы решить задачу на полный балл, можно заметить, что ответ будет иметь вид из последовательности нулей, за которой идёт последовательность единиц, так как если какая-то бактерия может выиграть, то все бактерии большего размера тоже могут выиграть. Это позволит написать решение при помощи двоичного поиска. Давайте двоичным поиском искать минимальный номер бактерии, которая может выиграть. Проверку того, может ли выиграть эта бактерия, будем делать моделированием. Тогда мы сократим количество запускаемых моделирований с n до $O(\log n)$, а общее решение будет иметь сложность $O(n \log n)$.

Но можно обойтись из без двоичного поиска. Все бактерии упорядочены по неубыванию размеров. Давайте для каждой бактерии проверим, сможет ли эта бактерия съесть следующую за ней бактерию в этом списке, если она уже съела все бактерии, которые находятся до неё.

Например, если размеры бактерий были $[1, 2, 2, 7, 8, 30, 40]$, то полученный ответ будет иметь вид $[0, 1, 0, 1, 0, 1, 1]$. Поясним, для каких бактерий в этом ответе записан 0. Это самая маленькая бактерия, вторая бактерия размером 2 (она не сможет съесть бактерию размером 7, т.к. если она съест всех перед ней, то её размер станет 5, а следующая бактерия имеет размер 7), бактерия размером 8 (если она съест всех перед ней, то её размер будет 20, а размер следующей бактерии равен 30). Но полученный массив не является правильным ответом, т.к. если для какой-то бактерии записано 1, а правее неё стоит 0, то в ходе дальнейшего поедания бактерий по возрастанию масс она не сможет дойти до конца).

Поэтому в этом массиве нужно найти последнее число 0 и обнулить все элементы до него. Пример решения на 100 баллов сложности $O(n)$.

```
n = int(input())
a = [int(input()) for i in range(n)]
ans = [0] * n
s = 0
for i in range(n - 1):
    if a[i] > a[0] and a[i] + s > a[i + 1]:
        ans[i] = 1
    else:
        ans[i] = 0
    s += a[i]
if a[n - 1] > a[0]:
    ans[n - 1] = 1
```

```
else:
    ans[n - 1] = 0
i = n - 1
while ans[i] == 1:
    i -= 1
while i >= 0:
    ans[i] = 0
    i -= 1
if n == 1:
    ans = [1]
for i in range(n):
    print(ans[i])
```

Отметим несколько сложных случаев. Если $n > 1$ и размеры всех бактерий равны, то в этом случае ответ состоит из одних нулей, т.к. ни одна бактерия не сможет съесть остальные. Это тест № 7, и это единственный случай, когда ответ состоит из одних нулей (даже самая большая бактерия не может выиграть).

Но если $n = 1$, то это вырожденный случай, когда ответ состоит из одних единиц и может выиграть даже самая маленькая бактерия, потому что других бактерий нет. Это тест № 6. Возможно, обработка этих случаев потребует отдельного if в решении.