Всероссийская олимпиада школьников по информатике, 2017/18 уч. год
Первый (школьный) этап, г. Москва
Решения заданий для 9–11 классов

Задача 1. Спиннеры

Автор задачи — Михаил Прохоров

Спиннер – модная игрушка с подшипником в основании, к которому прикреплены лопасти. Афанасий открыл бизнес по производству спиннеров. Он выяснил, что за спиннер, у которого N лопастей, покупатели готовы платить $A + B \times N$ рублей, но при этом покупатель не станет покупать спиннер, если его цена будет выше C рублей. Определите максимальное число лопастей спиннера, который согласится приобрести покупатель.

Программа получает на вход три числа A, B, C (стоимость основания спиннера, стоимость одной лопасти и максимальная стоимость всего спиннера). Все числа – целые положительные, не превосходящие 2×10^9, при этом $A \le C$.

Программа должна вывести одно число – максимальное число лопастей спиннера.

Пример входных и выходных данных

Ввод	Вывод	Примечание
20 10 55	3	Спиннер с 3 лопастями будет стоить 50 рублей, а с 4 лопастями – 60 рублей. Максимальная возможная стоимость спиннера – 55 рублей, поэтому максимальное число лопастей равно 3.

Система оценивания

Решение, правильно работающее только для случаев, когда все входные числа не превосходят 100, будет оцениваться в 60 баллов.

Решение

По условию задачи $A + B \times N \le C$, откуда $N \le (C - A) / B$. Программа должна вывести целочисленное частное от деления $(C - A) / B$, то есть округлить результат вниз до целого числа. Для этого в языке Pascal используется операция div, в языке Python — операция «//», в языках C и C++ — операция «/» для целочисленных операндов. Пример полного решения.

```
a = int(input())
b = int(input())
c = int(input())
n = (c - a) // b
print(n)
```

60 баллов набирали решения, в которых значение N увеличивалось в цикле на 1, пока выполнялось условие $A + B \times N \le C$. Пример решения на 60 баллов.

```
a = int(input())
b = int(input())
c = int(input())
n = 0
while a + b * n <= c:
    n = n + 1
print(n - 1)
```

Задача 2. Снова спиннеры

Денис тоже решил заняться производством и продажей спиннеров, но он считает, что у спиннера может быть только три или четыре лопасти. У него есть ровно M лопастей, которые он может прикреплять к основаниям, и неограниченный запас оснований. Он хочет изготовить несколько трёхлопастных и несколько четырёхлопастных спиннеров так, чтобы использовать все M лопастей. Определите, сколько спиннеров каждого вида он должен произвести.

Программа получает на вход одно целое положительное число M, не превосходящее 2×10^9, – количество лопастей, которое есть у Дениса.

Программа должна вывести два целых числа – количество спиннеров с 3 лопастями и количество спиннеров с 4 лопастями, которые должен произвести Денис. Если у задачи есть несколько решений, нужно вывести любое из них. Если Денис не может использовать ровно M лопастей для производства спиннеров, программа должна вывести два числа 0.

Примеры входных и выходных данных

Ввод	Вывод	Примечание
10	2 1	10 = 3 × 2 + 4 × 1
1	0 0	Невозможно произвести спиннеры так, чтобы суммарное число лопастей было равно 1.

Система оценивания

Решение, правильно работающее только для случаев, когда M не превосходит 100, будет оцениваться в 60 баллов.

Решение

Задача имеет решения для любого M, кроме 1, 2, 5. Заметим, что если число M делится на 3, то можно изготовить только трёхлопастные спинеры, их количество будет равно M / 3. Если M даёт остаток 1 при делении на 3, то необходимо изготовить как минимум один четырёхлопастный спиннер. Если M даёт остаток 2 при делении на 3, то понадобится минимум 2 трёхлопастных спиннера. Таким образом, можно считать, что число четырёхлопастных спиннеров равно остатку от деления M на 3, запишем это количество в переменную n_4. Для нахождения остатка от деления в языке Pascal используется операция mod, в языках C, C++ и Python операция «%». Оставшиеся лопасти будем использовать для производства трёхлопастных спиннеров, их количество будет равно $(M - 4\,n_4) / 3$. Если при этом число трёхлопастных спиннеров получилось отрицательным, то задача решения не имеет. Пример решения на языке Python.

```
m = int(input())
n4 = m % 3
n3 = (m - 4 * n4) // 3
if n3 >= 0:
    print(n3)
    print(n4)
else:
    print(0)
    print(0)
```

Задача 3. Не про спиннеры

Автор задачи — Александр Серянин

Саша совсем не любит спиннеры, поэтому он рисует в тетрадке. Он взял тетрадный лист из $N \times M$ клеточек и пронумеровал все клетки различными числами. Теперь ему стало интересно, сколько различных прямоугольников он может вырезать из этого листа бумаги по границам клеточек.

Программа получает на вход два числа N и M – размеры исходного листа. Все числа – целые положительные, не превосходящие 75000.

Программа должна вывести одно число – количество прямоугольников, которые можно вырезать из данного листа бумаги (весь лист целиком также считается одним из возможных прямоугольников).

Примеры входных и выходных данных

Ввод	Вывод	Примеры вырезанных прямоугольников
2 2	9	
3 1	6	

Система оценивания

Решение, правильно работающее только для случаев, когда все входные числа не превосходят 10, будет оцениваться в 40 баллов.

Решение, правильно работающее только для случаев, когда все входные числа не превосходят 200, будет оцениваться в 70 баллов.

Решение

Пронумеруем столбцы листа числами от 1 до N, строки листа числами от 1 до M. Пусть клетка, находящаяся в одном углу прямоугольника имеет координаты x_1, y_1, а противоположная клетка — координаты x_2, y_2. Тогда выполнены неравенства $1 \le x_1 \le x_2 \le N$, $1 \le y_1 \le y_2 \le M$. Задача сводится к подсчёту количества четвёрок чисел x_1, y_1, x_2, y_2, удовлетворяющих этим неравенствам.

Если это делать при помощи четырёх циклов, то получится решение сложности $O(N^2M^2)$, которое набирает 40 баллов. Пример такого решения.

```
n = int(input())
m = int(input())
ans = 0
for x1 in range(n):
    for x2 in range(x1, n):
        for y1 in range(m):
            for y2 in range(y1, m):
                ans += 1
print(ans)
```

Для получения больших баллов необходимо оптимизировать это решение. Например, можно заметить, что задачу можно решать независимо по каждой из двух осей координат, то есть нужно подсчитать количество подходящих пар x_1, x_2 и количество подходящих пар y_1, y_2

и перемножить два найденных числа. Такое решение будет иметь сложность $O(N^2 + M^2)$ и набирает 70 баллов. Пример такого решения:

```
n = int(input())
m = int(input())
ans_x = 0
ans_y = 0
for x1 in range(n):
    for x2 in range(x1, n):
        ans_x += 1
for y1 in range(m):
    for y2 in range(y1, m):
        ans_y += 1
print(ans_x * ans_y)
```

Для дальнейшего улучшения решения заметим, что во вложенных циклах к числу прибавляется 1, поэтому можно оба вложенных цикла заменить на увеличение переменной на число, равное количеству итераций вложенного цикла. Такое решение будет иметь сложность $O(N + M)$ и набирает 100 баллов. Пример такого решения:

```
n = int(input())
m = int(input())
ans_x = 0
ans_y = 0
for x1 in range(n):
    ans_x += n - x1
for y1 in range(m):
    ans_y += m - y1
print(ans_x * ans_y)
```

Это решение уже набирает полный балл, но и его можно упростить, если заметить, что в первом цикле суммируются числа $n + (n – 1) + … + 1$, и эта сумма равна $n (n + 1) / 2$. Аналогично, сумма чисел во втором цикле равна $m (m + 1) / 2$. Можно перемножить эти два числа, получится решение сложности O(1). Пример такого решения:

```
n = int(input())
m = int(input())
print(n * (n + 1) // 2 * m * (m + 1) // 2)
```

Заметим, что для хранения ответа в решениях на языках Pascal, C, C++ необходимо использовать 64-битную целочисленную арифметику, так 32-битного целого числа не хватит для представления ответа, если значения N и M больше 200. Поэтому даже эффективные решения на языках Pascal, C, C++, использующие 32-битные целые числа, набирали 70 баллов.

Задача 4. Плацкартный вагон

Автор задачи — Владимир Пугнин

В плацкартном вагоне 54 места, пронумерованных числами от 1 до 54. Вагон разбит на 9 купе. Первые 36 мест расположены по левую сторону от прохода, места 1–4 находятся в первом купе, места 5–8 – во втором и т. д. В девятом купе находятся места с номерами 33–36. По правую сторону от прохода находятся боковые места, их номера от 37 до 54, причём они нумеруются в противоположном направлении: места 37 и 38 находятся напротив девятого купе, а места 53 и 54 – напротив первого. Ниже приведена схема всех мест в вагоне.

1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36
53	54	51	52	49	50	47	48	45	46	43	44	41	42	39	40	37	38

Группа школьников едет на олимпиаду и будет всю дорогу крутить спиннеры. Поэтому им нужно купить места в нескольких подряд идущих купе вместе с прилегающими боковыми местами. Даны номера свободных мест в поезде. Определите, какое наибольшее число подряд идущих купе полностью свободны.

Программа получает на вход число N – количество свободных мест в вагоне ($0 \le N \le 54$). Следующие N строк содержат номера свободных мест – различные числа от 1 до 54 в произвольном порядке, по одному числу в строке.

Программа должна вывести одно целое число – максимальное число подряд идущих свободных купе (купе – 4 места слева от прохода и 2 боковых места) в этом вагоне.

Примеры входных и выходных данных

Ввод	**Вывод**	**Примечание**
12 5 6 3 4 8 7 51 9 10 54 49 52	1	Свободно одно купе с местами 5, 6, 7, 8, 51, 52.
1 1	0	В вагоне только одно свободное место, поэтому свободных купе нет совсем.

Система оценивания

Решение, правильно работающее только для случаев, когда ответом является число 0 или 1, будет оцениваться в 40 баллов.

Решение

Трудность в задаче может представлять только реализация алгоритма решения.

Сначала нужно научиться по номеру свободного места определять номер купе, в котором находится данное место. В приведенном ниже решении это делает функция section, которая рассматривает два случая: номер места не превосходит 36 и номер места больше 36.

Создаётся массив count, где count[i] будет равно числу свободных мест в купе номер i. Этот массив заполняется нулями, при считывании свободного места определяется номер купе, в котором находится это место, и соответствующий элемент массива увеличивается на 1.

После окончания считывания данных полностью свободные купе — это те купе, для которых count[i] будет равно 6. Необходимо в массиве count найти самое большое число подряд идущих элементов, равных 6, что является стандартным алгоритмом обработки массивов.

Пример решения на языке Python.

```
def section(k):
    if k <= 36:
        return (k - 1) // 4
    else:
        return 8 - (k - 37) // 2

count = [0] * 9
n = int(input())
for i in range(n):
    count[section(int(input()))] += 1
ans = 0
curr = 0
for i in range(9):
    if count[i] == 6:
        curr += 1
        ans = max(ans, curr)
    else:
        curr = 0
print(ans)
```

Задача 5. Кинотеатр

В первом ряду кинотеатра $N + 2$ мест, крайние места заняты персоналом кинотеатра, но N мест посередине свободно. K школьников входят в зрительный зал по очереди, и, конечно же, каждый школьник достаёт спиннер и начинает его крутить до начала сеанса. Поэтому каждый школьник выбирает себе место как можно дальше от уже занятых мест. А именно, школьник находит самый большой свободный участок в ряду (любой, если таких несколько) и садится посередине него. Если число свободных мест на этом участке было нечётно, то школьник садится точно посередине участка, тогда слева и справа от него остаётся поровну свободных мест. Если же это число чётно, то школьник выбирает одно из двух свободных мест посередине, тогда с одной стороны от школьника будет на одно свободное место больше, чем с другой стороны.

По данным числам N и K определите, сколько мест осталось свободными с двух сторон от школьника, который занял место последним (K-м по счёту).

Программа получает на вход два целых числа N и K, $1 \le K \le N \le 10^{18}$, и должна вывести два целых числа **в порядке неубывания** – количество свободных мест с двух сторон от школьника, который последним занял место в ряду.

Примеры входных и выходных данных

Ввод	**Вывод**	**Примечание**
10 1	4 5	В зале 10 свободных мест, первый школьник сел посередине, с одной стороны от него 4 места, с другой стороны – 5.
10 2	2 2	Второй вошедший в зал школьник садится посередине группы из 5 свободных мест, с каждой стороны от него остаётся по 2 свободных места.
10 3	1 2	После того, как два школьника сели на места, в зале остались группы свободных мест из 4, 2, 2 мест. Третий школьник садится посередине группы из 4 мест, поэтому с одной стороны от него 1 место, с другой стороны – 2 места.

Система оценивания

Решение, правильно работающее только для случаев, когда N не превосходит 100, будет оцениваться в 30 баллов.

Решение, правильно работающее только для случаев, когда N не превосходит 10^5, будет оцениваться в 60 баллов.

Решение

На 30 баллов можно было реализовать решение сложности $O(N\,K)$, в котором в массиве, заполненном числами 0 или 1 будем отмечать, какие места свободны, а какие — заняты. Далее моделируется процесс выбора места: в массиве находится самый длинный непрерывный участок из 0, и элемент посередине этого участка меняется на 1. Приводить пример такого решения не будем.

Для решения на 60 баллов необходимо хранить не схему размещения всех свободных мест в ряду, а только множество непрерывных участков из свободных мест. Например, если $N = 10$, то в самом начале есть один свободный участок длины 10. Первый приходящий человек садится посередине этого участка, поэтому вместо одного участка длины 10 в множество добавляются участки длины 5 и 4. Второй пришедший садится посередине

участка длиной 5, который разбивается на два участка длиной 2 и 2 и т. д. На каждом шаге из множества выбирается самый большой элемент и вместо него в множество добавляется два меньших элемента. При этом необходимо уметь быстро находить в множестве наибольший элемент и добавлять в множество новые элементы, для этого можно использовать структуру данных multiset языка C++. Такое решение будет иметь сложность $O(K \log K)$. Приведем пример такого решения (60 баллов).

```
#include<iostream>
#include<set>

using namespace std;

long long pop(multiset<long long> & s)
{
    auto it = s.end();
    --it;
    int ans = *it;
    s.erase(it);
    return ans;
}

int main()
{
    long long n, k;
    cin >> n >> k;
    multiset<long long> s;
    s.insert(n);
    long long ans1, ans2;
    for (long long i = 0; i < k; ++i)
    {
        n = pop(s);
        ans1 = (n - 1) / 2;
        ans2 = n - 1 - ans1;
        s.insert(ans1);
        s.insert(ans2);
    }
    cout << ans1 << endl;
    cout << ans2 << endl;
}
```

Для полного решения на 100 баллов заметим, что после выполнения большого числа операций множество свободных участков будет содержать большое число одинаковых значений, так как если два участка одинаковой длины поделить на части, то результат тоже будет одинаковой длины. Поэтому в множестве длин участков можно хранить не все участки равной длины, а для каждой имеющейся длины участка запоминать, сколько существует в настоящий момент участков такой длины. Тогда все участки равной длины будут

обрабатываться не по одному, а все вместе.

В приведенном ниже решении для этого используется словарь (ассоциативный массив) d, у которого ключ — это длина участка, а значение — количество участков заданной длины. Инициализация массива: d[n] = 1, то есть в самом начале есть один участок длины n.

```
n = int(input())
k = int(input())
d = dict()
d[n] = 1
while k > 0:
    m = max(d.keys())
    count = min(k, d[m])
    del d[m]
    k -= count
    ans1 = (m - 1) // 2
    ans2 = m - 1 - ans1
    d[ans1] = d.get(ans1, 0) + count
    d[ans2] = d.get(ans2, 0) + count
print(ans1)
print(ans2)
```

Без пояснений приведём пример, вероятно, самого короткого известного решения по этой задаче. Это решение было найдено участниками олимпиады, затем немного упрощено методической комиссией. В решении используется тот факт, что длины участков сокращаются в два раза, а количество участков такой длины — удваивается.

```
n = int(input())
k = int(input())
while k != 1:
    n = (n - k % 2) // 2
    k //= 2
print((n - 1) // 2, n // 2)
```