A. *Новый телефон*

Если Петя останется работать программистом, то через три месяца у него будет $3 \cdot X$ рублей.

Если Петя пойдет учиться на водителя троллейбуса, то за первый месяц он получит стипендию в размере A рублей, а за два месяца работы он получит зарплату в размере $2 \cdot B$ рублей, итого через три месяца у Пети будет $A + 2 \cdot B$ рублей.

Из этих двух вариантов необходимо выбрать тот, при котором Петя получит больше денег через три месяца, то есть ответом на задачу будет

$$\max(3 \cdot X, A + 2 \cdot B)$$

```
X = int(input())
A = int(input())
B = int(input())

print(max(3 * X, A + 2 * B))
```

B. *Физкультура*

В этой задаче можно просто промоделировать счёт, который должны вести дети и одновременно с этим пройтись по массиву и отметить сколько его элементов не совпали с правильными. Пример решения на языке `Python` приведён ниже.

```
N = int(input())
x = []
for i in range(N):
    x.append(int(input()))

ans = 0
for i in range(len(x)):
    if x[i] != 2 - (i + 1) % 2:
        ans += 1
print(ans)
```

C. *Три гирьки*

Без ограничения общности будем считать, что мы будем взвешивать груз, ставя его на левую чашку весов. Тогда если гири на левой и правой чашках весов уравновешивают груз, то вес груза равен весу всех гирек, стоящих на правой чашке, минус вес гирек, стоящих на левой чашке. Другими словами, гирька учитывается со знаком плюс, т.е. с коэффициентом $+1$, если стоит на правой чашке весов, со знаком минус, т.е. с коэффициентом -1, если на левой чашке, и не учитывается, т.н. учитывается с коэффициентом 0, если гирька не стоит на весах.

Это соображение позволяет перебрать все возможные расстановки гирек и записать в массив/список соответствующие веса грузов.

Для окончания решения достаточно удалить из этого массива/списка все неположительные числа, после чего найти количество различных чисел. Это можно сделать, например, с помощью сортировки и последующего прохода по массиву/списку либо добавить элементы во встроенную в язык программирования структуру данных (например, `set`). Также, учитывая, что суммарный вес гирек не превосходит 60, можно было завести булев массив на 61 элемент и отмечать в нём, какие из весов могли бы быть достигнуты.

Ниже приведены примеры решения с сортировкой, с булевым массивом и использованием множества соответственно.

```
A = int(input())
B = int(input())
C = int(input())

vs = []

for aa in range(-A, A + 1, A):
    for bb in range(-B, B + 1, B):
        for cc in range(-C, C + 1, C):
            ss = AA + BB + CC
            if ss > 0:
                vs.append(ss)

vs.sort()
ans = 1
for i in range(1, len(vs)):
    if (vs[i] != vs[i-1]):
        ans += 1
print(ans)
```

```
A = int(input())
B = int(input())
C = int(input())

w = [False] * (A + B + C + 1)
for a in -1, 0, 1:
    for b in -1, 0, 1:
        for c in -1, 0, 1:
            w[abs(a * A + b * B + c * C)] = True

print(w[1:].count(True))
```

```
import sys

a = int(input())
b = int(input())
c = int(input())

s = set()
for aa in -a, 0, a:
  for bb in -b, 0, b:
    for cc in -c, 0, c:
      if aa + bb + cc > 0:
        s |= {aa + bb + cc}

print(len(s))
```

D. *Классический сюжет*

В этой задаче требовалось разбить последовательность на кусочки по два или три одинаковых элемента либо сообщить, что это невозможно. В первую очередь разобьём последовательность на максимальные по включению куски, состоящие из одинаковых чисел. Очевидно, что для решения задачи необходимо и достаточно подразбить на кусочки длин 2 и/или 3 именно такие куски.

Очевидно, подразбить кусок длины 1 невозможно; вследствие этого, если хотя бы один кусок имеет длину 1, то нужно вывести -1.

С другой стороны, любой кусок длины L, $L \geqslant 2$ подразбить можно. Один из, по-видимому, наиболее простых способов подразбиения следующий:

- если L чётно, то нужно подразбить кусок на $L/2$ кусков длины 2;
- если L нечётно, то нужно подразбить кусок на один кусок длины 3 и $(L-1)/2$ кусков длины 2.

Возможная реализация на языке Python приведена ниже.

```
import sys

n = int(input())

a = []
for i in range(n):
    a.append(input())

b = []

i = 0
while i < n:
    ni = i + 1
    while ni < n and a[i] == a[ni]:
        ni += 1
    b.append((a[i], ni - i))
    if ni - i == 1:
        print(-1)
        sys.exit()
    i = ni

for num, kol in b:
    while kol > 0:
        ckol = 2 + (kol % 2)
        print(ckol, ' '.join([num] * ckol))
        kol -= ckol
```

Другой вариант реализации решения — с простыми структурами данных (но с двумя проходами по исходному массиву) и без необходимости использования модуля sys.

```
N = int(input())
x = []
for i in range(N):
    x.append(int(input()))

has_solution = True
curr_len = 1
for i in range(1, len(x)):
    if x[i] == x[i - 1]:
        curr_len += 1
    else:
        if curr_len == 1:
            has_solution = False
        curr_len = 1
if curr_len == 1:
    has_solution = False

if not has_solution:
    print(-1)
else:
    curr_len = 1
    for i in range(1, len(x)):
        if x[i] == x[i - 1]:
            curr_len += 1
        else:
            if curr_len % 2 == 1:
                print(3, x[i - 1], x[i - 1], x[i - 1])
                curr_len -= 3
            for k in range(curr_len // 2):
                print(2, x[i - 1], x[i - 1])
            curr_len = 1
    if curr_len % 2 == 1:
        print(3, x[i - 1], x[i - 1], x[i - 1])
        curr_len -= 3
    for k in range(curr_len // 2):
        print(2, x[i - 1], x[i - 1])
```

E. *Лось Валера*

Представим себе ряд натуральных чисел, из которых надо «собрать» требуемую перестановку: $1\ 2\ 3\ 4 \ldots N$

Будем строить перестановку последовательно, слева направо, вместе с чтением строки из символов со знаками неравенств.

В процессе построения перестановки надо помнить два числа: левую (`L`) и правую (`R`) границы. Левая граница в начале равна 1, правая равна N. Они будут обладать следующим свойством: на каждом этапе построения перестановки в ней уже есть все числа, меньше L и все числа, больше R.

Далее выполняем следующую процедуру: если при проходе строки из неравенств слева направо мы видим знак `>`, то в перестановку ставим число `R` и уменьшаем его на 1, в противном случае ставим число `L` и увеличиваем его на 1.

После окончания цикла значения `R` и `L` будут равны, и последним элементом перестановки можно сделать любое из них.

Для доказательства корректности такого алгоритма переформулируем свойства чисел L и R: число L меньше всех пока не поставленных чисел перестановки, число R больше всех пока не поставленных чисел перестановки.

Таким образом мы всегда можем расставить числа в такой цепочке неравенств. Ниже приведён пример программы на языке `Python`.

```
s = input()
left = 1
right = len(s) + 1
for i in range(len(s)):
    if s[i] == '<':
        print(left)
        left += 1
    else:
        print(right)
        right -= 1
print(left)
```

F. *Сумма цифр 2: отсутствие легенды*

Выпишем несколько первых чисел с суммой цифр 2, идущих подряд:

2
11
20
101
110
200
1001
1010
1100
2000
10001
10010
10100
11000
20000
100001
100010
...

Внимательно посмотрев на последовательность, можно заметить, что перебирать числа с суммой цифр 2 в порядке возрастания можно следующим образом.

Пусть разряды числа нумеруются слева направо, начиная с нуля. Тогда каждое число с суммой цифр 2 задаётся целыми неотрицательными числами i, j, $i \geqslant j \geqslant 0$, где i и j — это разряды, в которых находятся единицы.

Например, для числа 10010 $i = 4$, $j = 1$, а для числа 1100000 $i = 6$, $j = 5$. Обратите внимание: i и j могут совпадать!

Например, для числа 2000 $i = j = 3$.

Будем перебирать вместо самих чисел подходящие пары (i, j). В каком же порядке их перебрать? Очевидно, что при движении вдоль вышеприведенной последовательности i не убывает (и это не случайно, ибо $i+1$ есть длина числа), а при фиксированном i, j пробегает все значения от нуля до i.

Более того, отметим, что по числам i, j можно напечатать на экране число, не вычисляя его в явном виде!

Например, если $i > j$, то достаточно:

- напечатать 1;
- напечатать $i - j - 1$ нулей;
- напечатать еще раз 1;
- напечатать j нулей.

Если же $i = j$, то достаточно напечатать цифру 2, а затем i нулей. Все эти соображения приводят нас к следующей программе:

```
import sys

n = int(input())
for i in range(0, n + 1):
    for j in range(0, i + 1):
        n -= 1
        if n == 0:
            s = []
            if i == j:
                s.append('2')
            else:
                s.append('1')
                for k in range(j + 1, i):
                    s.append('0')
                s.append('1')
            for k in range(0, j):
                s.append('0')
            print(''.join(s))
            sys.exit(0)
```

Ниже приводится совсем другой способ решения этой задачи. Попробуйте самостоятельно разобраться, как работает эта программа. Может быть, вам пригодится табличка, где числа пронумерованы и разбиты на группы по количеству разрядов:

1. 2

2. 11
3. 20

4. 101
5. 110
6. 200

7. 1001
8. 1010
9. 1100
10. 2000

11. 10001
12. 10010
13. 10100
14. 11000
15. 20000

```
N = int(input())
L = 1
S = 0
while S + L < N:
    S += L
    L += 1

print(10 ** (L - 1) + 10 ** (N - S - 1))
```

G. *Игра в шпионов*

Предложим Маше следующий алгоритм. «Запомним» пока первое число, сообщённое Ваней, отметив про себя, что оно равно количеству единиц в его пароле.

Далее, начиная со второго числа, будем смотреть на соотношение текущего и предыдущего чисел. Ясно, что они не могут совпадать. Также ясно, что следующее число может отличаться от предыдущего только на 1.

Если оно больше на единицу (то есть в шаблоне `100..000` на одно совпадение больше, чем в шаблоне `000..000`), значит первый символ пароля это «1», а если меньше на единицу, то «0».

Действуя аналогичным образом, мы получаем все символы Ваниного пароля, кроме последнего. Тут настало время вспомнить первое число! Если мы считали, сколько к этому моменту в Ванином пароле единиц, то последний символ его пароля угадать несложно.

Ниже приведено решение на языке `Python`.

```
N = int(input())
x = []
for i in range(N):
    x.append(int(input()))

answer = []
ones = x[0]
cnt_ones = 0

for i in range(1, len(x)):
    if x[i] > x[i - 1]:
        answer.append(0)
    else:
        cnt_ones += 1
        answer.append(1)

if cnt_ones < ones:
    ans.append(1)
else:
    ans.append(0)

for i in range(len(answer)):
    print(answer[i], end = '')
```

Отметим, что при условии, что последовательность, сообщённая Ваней не имеет ошибок, Маша всегда может однозначно восстановить его пароль. Количество несовпадений, упоминавшееся в условии, называется расстоянием Хэмминга и является элементарным способом вычислять расстояние между строками, в данном случае одинаковой длины.

H. *Имена на карточках*

Мы может отрезать i букв в начале карточки и j букв в конце карточки. Чтобы получилась не пустая карточка нужно чтобы выполнялось неравенство $i+j < n$, где n — длина слова, которое принтер печатает на карточках. Переберём все значения i и j от 0 до $n-1$. Для тех пар i, j для которых выполнено неравенство $i + j < n$ сохраним соответствующее данной обрезанной карточке слово в массив `all_names`.

Чтобы найти ответ для данной задачи нужно подсчитать, сколько различных слов записано в массиве `all_names`. Для этого отсортируем этот массив каким-нибудь способом, например, пузырьком или встроенной сортировкой. После этого все одинаковые слова будут идти в этом массиве подряд, поэтому если пройти по массиву и посчитать только те слова, у которых слева не стоит такое же слово, то получим ответ на нашу задачу.

Можно было обойтись и без сортировки, на каждом шагу проверяя, есть ли уже такое слово в массиве `all_names`.

Ниже приведены оба решения на языке `Python`:

```
s = input()
n = len(s)
all_names = []
for i in range(n):
    for j in range(i, n):
        all_names.append(s[i:(j + 1)])
L = len(all_names)
for i in range(L):
    for j in range(1, L):
        if (all_names[j] > all_names[j - 1]):
            all_names[j - 1], all_names[j] = all_names[j], all_names[j - 1]

ans = 1
for i in range(1, L):
    if (all_names[i] != all_names[i - 1]):
        ans += 1
print(ans)
```

```
s = input()

all_names = []
for start in range(len(s)):
    for fin in range(start + 1, len(s) + 1):
        if s[start:fin] not in all_names:
            all_names.append(s[start:fin])

print(len(all_names))
```

I. *Игра в камушки*

Давайте заведем массив dp размера $n + 1$. Если игрок, которому нужно делать ход в тот момент, когда в кучке лежит i камней выигрывает при правильной стратегии, то положим $dp[i] = 1$, в противном случае $dp[i] = 0$. Тогда, если $dp[n] = 1$, то выигрывает первый игрок, в противном случае выигрывает второй игрок.

Но как заполнить массив dp? Будем это делать по возрастанию числа i. Так как при $i = 0$ игрок проигрывает, то $dp[0] = 0$. Далее перебираем все значения i последовательно от 1 до n.

- Если мы можем взять из кучки один камень и получить позицию, в которой наш соперник проиграет, то значит мы сможем выиграть, то есть если $dp[i - 1] = 0$, то $dp[i] = 1$.
- Если мы можем взять из кучки два камня и получить позицию, в которой наш соперник проиграет, то значит мы сможем выиграть, то есть если $i \geq 2$ и $dp[i - 2] = 0$, то $dp[i] = 1$.
- Если мы можем взять из кучки k камней и получить позицию, в которой наш соперник проиграет, то значит мы сможем выиграть, то есть если $i \geq k$ и $dp[i - k] = 0$, то $dp[i] = 1$.
- Если ни одно из трёх условий не выполняется, то $dp[i] = 0$.

Ниже приведено решение задачи на языке `Python`:

```
n = int(input())
k = int(input())

d = [0] * (n + 1)

for i in range(n + 1):
    for j in [1, 2, k]:
        if (i >= j and d[i - j] == 0):
            d[i] = 1

print(2 - d[n])
```