Задача A. Длинное и короткое

Определим массив строк `words[n]`, в котором будем хранить считанные слова из тетради Даши Карамелькиной. Отсортируем его по длине строк с помощью команды:

```
words.sort(key = len, reverse = True).
```

После этого самое длинное слово будет в элементе $words[0]$, а самое короткое — в $words[n-1]$.

Задача B. Наименьшее число

Подзадача 1. Если искомое число имеет небольшую сумму цифр n, то будем перебирать все натуральные числа, начиная с 1, пока не встретим число с заданной суммой цифр n. Первое встретившееся число, очевидно, будет требуемым.

Подзадача 2. Заметим, что требуемое число будет тем меньше, чем меньше цифры в *старших* разрядах. Расставим цифры 9 в младших разрядах нашего числа. Максимально возможное количество девяток равно целой части числа $k = n / 9$. Разность $m = n - k$ равна сумме цифр в оставшихся разрядах и она меньше 9. Другими словами, ненулевое число $m = n - k$ совпадает с цифрой старшего разряда искомого числа. Таким образом, искомое число имеет вид $\overline{m99\ldots9}$, где цифра 9 записана k раз.

Задача C. Соревнования по прыжкам

Мы поддерживаем интервал $[l, r]$, который представляет наименьшее и наибольшее значения, которые мы можем достичь из текущих позиций. Изначально, мы устанавливаем $l = r = a_0$, поскольку начинаем с позиции a_0.

Для каждой позиции a_i (где $i > 0$) мы проверяем, находится ли a_i в интервале, расширенном на K, то есть проверяем условие:

$$l - K \leqslant a_i \leqslant r + K.$$

Если это условие выполняется, это означает, что мы можем достичь позиции i. В этом случае мы обновляем интервал $[l, r]$, чтобы включить значение a_i, то есть:

$$l = \min(l, a_i),$$

$$r = \max(r, a_i).$$

Если a_i не находится в этом интервале, мы не можем достичь позиции i, и интервал остается неизменным.

Решение проходит через массив один раз, поэтому временная сложность составляет $O(N)$.

Пример (из задачи): Для массива $a = [5, 4, 8, 7, 2]$ и $K = 2$ интервалы будут обновляться следующим образом:

- Начальный интервал: $[5, 5]$
- Позиция $a_1 = 4$: в интервале $[5 - 2, 5 + 2] = [3, 7]$, поэтому мы обновляем интервал до $[4, 5]$
- Позиция $a_2 = 8$: вне интервала $[4 - 2, 5 + 2] = [2, 7]$, мы не можем достичь этой позиции, интервал остается $[4, 5]$
- Позиция $a_3 = 7$: в интервале $[4 - 2, 5 + 2] = [2, 7]$, поэтому мы обновляем интервал до $[4, 7]$
- Позиция $a_4 = 2$: в интервале $[4 - 2, 7 + 2] = [2, 9]$, поэтому мы обновляем интервал до $[2, 7]$

В конце концов, мы выводим, какие позиции достижимы: позиции 0, 1, 3 и 4 достижимы, а позиция 2 — нет.

Задача D. Арифметические тройки

Для простоты, мы предполагаем, что $A \leqslant B$. (В противном случае, мы можем поменять A и B местами.)

Условие: Среди A, B и x, разность между первым и вторым по величине значениями равна разности между вторым и третьим по величине значениями.

Рассмотрим два случая: $A = B$ и $A < B$.

Если $A = B$

Условие выполняется тогда и только тогда, когда $x = A = B$. Таким образом, ответ равен 1.

Если $A < B$

Существуют три варианта расположения чисел A, B и x: $x < A < B$, $A < B < x$ и $A < x < B$. (Мы исключаем возможность того, что x равно либо A, либо B, в этом случае условие никогда не выполняется.)

- Если $x < A < B$, то тройка чисел x, A и B будет арифметической, если $B - A = A - x$, поэтому искомое значение x однозначно определяется как $x = 2A - B$.
- Аналогично, если $A < B < x$, подходящее x однозначно определяется как $x = 2B - A$.
- Если же $A < x < B$, то из условия $B - x = x - A$, находим $x = \frac{1}{2}(A + B)$, при этом число x будет целым тогда и только тогда, когда $A + B$ — чётное.

Резюме

Согласно вышеизложенному, ответ на эту задачу:

- 1, если $A = B$;
- 3, если $A \neq B$ и $A + B$ чётное;
- 2, если $A \neq B$ и $A + B$ нечётное.

Задача E. Квартирные обмены

Подзадача 1

Заметим, что применение ключа b сдвигает перестановку a на один элемент циклически вправо. Если применять k раз этот ключ, то массив a циклически сдвинеться на k вправо. Так как k может быть больше n, то заметим, что после применения n раз этого ключа массив примет исходный вид. Значит, сдвинем элементы вправо на $k \mod n$.

Подзадача 2

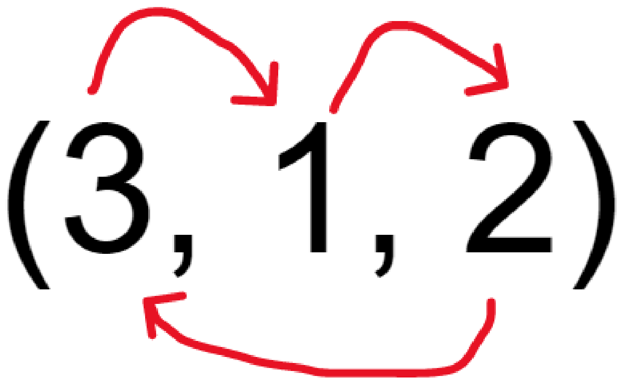
Заметим, что применение ключа b сдвигает перестановку a на t элементов циклически в право. Если применить этот ключ k раз, то массив a циклически сдвинется на $t \cdot k$ вправо. Так как $t \cdot k$ может быть больше n, то заметим, что после применения n раз этого ключа массив примет исходный вид. Значит, сдвинем элементы вправо на $(k \cdot t) \pmod n$.

Подзадача 3

Надо реализовать перемножение перестановок.

Подзадача 4

Рассмотрим умножение перестановки (a_1, a_2, a_3) на перестановку $g = (3, 1, 2)$ получим (a_3, a_1, a_2). Умножим на перестановку g еще раз, получим (a_2, a_3, a_1), проделаем то же самое третий раз и получим (a_1, a_2, a_3), массив принял исходный вид, то есть после трех перемножений перестановки массив принял исходный вид. Заметим, что элемент a_1 был на позициях $1 \to 2 \to 3 \to 1$, потом он снова будет перемещаться по данному циклу, то есть каждый элемент будет двигаться по определенному циклу. Рассмотрим цикл в перестановке g на рисунке.



Пусть t — это НОК всех длин циклов исходной перестановки-ключа b, тогда после перемножение t раз перестановки b получим исходную перестановку. Данный факт несложно доказывается. Значит, можно найти все длины циклов и посчитать НОК t, после чего перемножить перестановку b всего $k \mod t$ раз.

Входные данные позволяли перемножать перестановку до тех пор, пока не получим исходную, что значит, что асимптотика решения $O(n \cdot \min(k, t))$.

Подзадача 5

Для полного решения задачи заметим, что умножение перестановок ассоциативно: $(a(bc))_i = ((ab)c)_i$. Тогда найдем произведение k перестановок-ключей $\underbrace{b \cdot b \cdot \ldots \cdot b}_{k\text{раз}}$, пусть k четное число, тогда, вычислив $d = \underbrace{b \cdot b \cdot \ldots \cdot b}_{\frac{k}{2}\text{раз}}$, мы можем найти $\underbrace{b \cdot b \cdot \ldots \cdot b}_{k\text{раз}} = d \cdot d$. Если k нечетное, то проделаем то же самое с $k - 1$, а потом умножим результат на b. То есть можно вычислять перемножение k перестановок-ключей с помощью бинарного возведения в степень.

```python
def mult(a, b):
  res = [a[b[i] - 1] for i in range(len(b))]
  return res


def PowPermutation(a, h):
  tmp = [i+1 for i in range(len(a))]

  while h:
    if h % 2 == 0:
      a = mult(a, a)
      h //= 2
    else:
      tmp = mult(tmp, a)
      h -= 1
  return tmp

```

```

def main(n, a, b, k):
  return mult(a, PowPermutation(b, k))


n = int(input())
a = list(map(int, input().split()))
b = list(map(int, input().split()))
k = int(input())
print(*main(n, a, b, k))
```