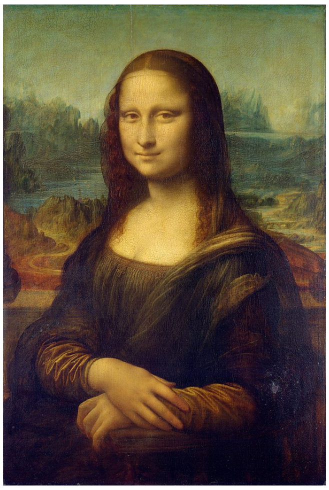
J: Mona Lisa

Time limit: 2 seconds



The Louvre museum hosts one of the most famous paintings ever made: Mona Lisa, painted by Leonardo da Vinci in the 16th century.

The painting is enclosed in a rock-solid glass chamber that can only be opened with 4 secret codes that need to be entered on 4 different keypads. The head of the museum thinks that this system is unbreakable, and your task is to prove her wrong.

To help you, a friend reverse-engineered the system. When a code (represented by a positive integer C) is entered on a keypad, the keypad sends the C-th value produced by a random number generator to a central computer. The central computer only considers the N least significant bits of the 4 pseudo-random values it receives from the 4 keypads. It computes their bitwise XOR (exclusive or), and opens the glass chamber if the result is 0. The pseudo-random number generator is described at the end of the problem statement.

Another friend found the pseudo-random seeds used by each keypad. With all this information, you think than you can retrieve the 4 secret codes unlocking Mona Lisa.

Input

The input comprises two lines, each consisting of integers separated with single spaces:

- The first line contains the integer N.
- The second line contains the four integer seeds.

Limits

- $1 \leqslant N \leqslant 50$;
- each seed is between 0 and $2^{64} - 1$.

Output

The output should consist of a single line, whose content is 4 integers, the 4 secret codes, separated with single spaces. Each code must be less than 100 000 000. It is guaranteed that at least one solution will exist. Multiple solutions may exist, in which case they will all be accepted.

Pseudo-Random Generator

The pseudo-random generator is described next in each programming language. You can expect that this pseudo-random generator is not biased in any way.

C/C++

```
typedef unsigned long long uint64;
uint64 state[2] = { seed, seed ^ 0x7263d9bd8409f526 };
uint64 xoroshiro128plus(uint64 s[2]) {
  uint64 s0 = s[0];
  uint64 s1 = s[1];
  uint64 result = s0 + s1;
  s1 ^= s0;
  s[0] = ((s0 << 55) | (s0 >> 9)) ^ s1 ^ (s1 << 14);
  s[1] = (s1 << 36) | (s1 >> 28);
  return result;
}
```

The *i*-th value of the pseudo-random sequence is the result of the *i*-th application of `xoroshiro128plus` on ‘`state`’.

Java

```
long[] state = { seed, seed ^ 0x7263d9bd8409f526L };
long xoroshiro128plus(long[] s) {
  long s0 = s[0];
  long s1 = s[1];
  long result = s0 + s1;
  s1 ^= s0;
  s[0] = ((s0 << 55) | (s0 >>> 9)) ^ s1 ^ (s1 << 14);
  s[1] = (s1 << 36) | (s1 >>> 28);
  return result;
}
```

The *i*-th value of the pseudo-random sequence is the result of the *i*-th application of `xoroshiro128plus` on ‘`state`’.

Python 2 / Python 3

```
state = [seed, seed ^ 0x7263d9bd8409f526]
def xoroshiro128plus(s):
  s0, s1 = s
  result = (s0 + s1) % 2**64
  s1 ^= s0;
  new_state = [(((s0 << 55) | (s0 >> 9)) ^ s1 ^ (s1 << 14)) % 2**64,
               ((s1 << 36) | (s1 >> 28)) % 2**64]
  return result, new_state
```

The following loop yields the pseudo-random sequence starting from its first value:

```
while True:
  result, state = xoroshiro128plus(state)
  yield result
```

Sample Input

```
50
3641603982383516983 4453636816169626440 8681964081858191799 1980241222855773941
```

Sample Output

```
287 17609 122886 59914
```