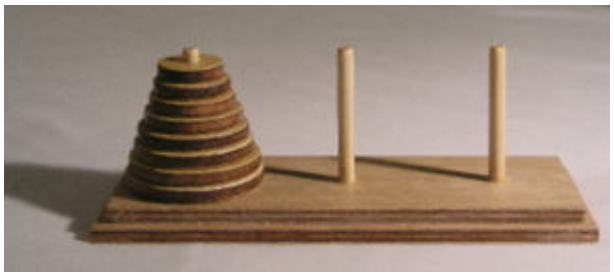
Tower of Hanoi P52637_en



In 1883, Edouard Lucas invented, or perhaps reinvented, one of the most popular puzzle of all time —Tower of Hanoi, as he called it— and that nowadays, it is already used as prototype of recursive algorithm. In fact, a lot of computing science textbooks, use it as instance to implement the idea of recursion in a program. These are the rules of the game:

- We have three poles (rods in the board game), that we will call A, B and C.
- We have n disks with a hole in the middle. The number of disks does not change while the puzzle is being solved.
- All the disks have different size.
- The disks are initially inserted in the pole A, in decreasing order from the ground. That is, the biggest one at the bottom to the smallest one at the top of the tower.
- The aim of the game is to move the whole tower from the A pole to one of the other two poles.
- Only one disk can be moved at a time, and it has to be one of the disks that are at the top of the stacks (that are being formed in the respectively poles during the development of the game) to other tower that has to be empty or to have in its top a bigger disk than the one that we are moving. That is, it is never allowed to place a disk over another one which is smaller.

Although, naturally, the best way to initiate youself into the secrets of this puzzle is to try to solve it with one of many and varied real models that are found in the shops of presents, if it is not possible, a good way to capture the essence of the game is to write a program that shows us in the screen a model of the Tower of Hanoi and that allow us to simulate the movement of the disks. But it is not what we are going to propose in this task. The next step would be to write a program that solve us the puzzle in the most efficient way. Neither it is the aim of this task. The followed aim is the implementation of an efficient algorithm to know the state of solution of the game. To make easier the task, we do not ask for the detail of the disks that there are in each pole, but only how many there are after of determinated movement, when the puzzle is solved following a specifically algorithm, that we describe below.

It is perfectly known, and it is very easy to prove, that the minimal number of movements of disks that you have to do to complete the change of the original tower to another different pole is $2^n - 1$, being n the number of disks. A very simple algorithm, but that do not allow us to solve the puzzle in this quantity of movemements is the following one: for the odd movements, we take the smallest disk (we say number 1) of the pole where it is inserted and it is placed in the following one inside the circle succession $ABCABC\ldots$; for the even movements, the only possible movement that does not involve the disk number 1 is done.

Input

The input file consists of a series of lines. Each one of them contains two integer numbers n, m: n is the number of disks and m, that will be in the interval $[0, 2^n - 1]$, will be the number of the last movement done. The file ends with a line that contains two zeros and that must not be processed.

Output

The output also consists of a series of lines, one for each line of the input file (except the last one). Each one of them will be formed by three integer numbers, that represent the number of disks of every pole A, B and C respectively, after the movement m, and remembering that those movements have been done according to the described algorithm.

Scoring

- **TestA:** **20 Points**

  100 test cases with $n \leq 16$.

- **TestB:** **50 Points**

  100 test cases with $n \leq 50$. Number m may not fit in an integer of 32 bits, therefore it will be necessary to use integers of 64 bits, as the type *long long* of C or C++.

- **TestC:** **30 Points**

  100 test cases with $n \leq 100$.

Sample input 1

```
1 0
1 1
2 0
2 1
2 2
3 0
3 1
3 2
3 3
3 4
3 5
3 6
3 7
0 0
```

Sample output 1

```
1 0 0
0 1 0
2 0 0
1 1 0
0 1 1
3 0 0
2 1 0
1 1 1
1 0 2
0 1 2
1 1 1
1 2 0
0 3 0
```

Sample input 2

```
3 5
8 45
39 437370956880
0 0
```

Sample output 2

```
1 1 1
4 2 2
17 12 10
```

Sample input 3

```
84 33405917778590387650216 67
73 7501655746272260576287
99 443387824463857406963274647 79
0 0
```

Sample output 3

```
20 39 25
26 27 20
33 30 36
```

Problem information

Author: Miguel Revilla
Translator: Carlos Molina

Generation: 2026-01-25T11:11:30.835Z