# ESERCIZIO 1

There are a lot of great puzzles for programming job interviews. My favorite one is also known as one of the favorites among Google recruiters:

*You work in a 100 floor building and you get 2 identical eggs. You need to figure out the highest floor an egg can be dropped without breaking. Find an algorithm that is minimizing number of throws in the worst-case scenario.*

We can make a few assumptions:

* If an egg doesn’t break when dropped from some floor, then it will not break when dropped from any lower floors.
* An egg that survives a fall can be used again.
* A broken egg must be discarded.
* The effect of a fall is the same for all eggs.
* If an egg breaks when dropped, then it would break if dropped from a higher floor.

# One Egg

Whilst it’s not strictly part of the puzzle, let’s first imagine what we’d do if we had only one egg.

Once this egg is broken, that’s it, no more egg. So, we really have no other choice but to start at floor 1. If it survives, great, we go up to floor 2 and try again, then floor 3 … all the way up the building; one floor at a time. Eventually the egg will break\* and we’ll have a solution. For example, if it breaks on floor 57, we know that the highest floor that an egg can withstand a drop from is floor 56.

# Many Eggs

At the other extreme, what if we had an infinite number of eggs? (Or at least as many eggs as we need). What would our strategy be here? In this case we’d use one of a programmer’s favorite tools, the binary tree.

First we’d go to floor 50 and drop an egg. It either breaks, or it does not. The outcome of this drop instantly cuts our problem in half. If it breaks, we know the solution lives in the bottom half of the building (floor 1 – floor 49). If it survives, we know the solution is in the top half of the building (floor 51 – 100). On each drop, we keep dividing the problem in half and half again until we get to our solution.

The mathematicians in the audience will quickly see that the number of drops required for this solution is log2 n, where n is the number of floors of the building.

Because this building does not have a number of floors equal to a round number power of two, we need to round up to number of drops to get seven ( log2 100 = 6.644 )

(Using seven drops, we could solve this problem any building up to 128 floors. Eight drops would allow us to solve the puzzle for a building with twice the height at 256 floors …)

Depending on the final answer, the actual number of eggs broken using a binary search will vary. At worst it will be seven eggs (if the floor is actually floor 1, since every drop will break the egg). At best it will be no eggs (if the actually answer is floor 100, since the egg will survive every drop made).

# Back to two eggs

OK, let’s get back to the original two egg problem. As we’ve seen from above, the worst case using a binary search would break seven eggs; not acceptable when we only have two eggs.

# Intuitive answer

This way, our first egg should be used to split the 100 floors range into smaller ranges as efficiently as possible. Thus, an intuitive and popular answer is to throw the first egg from 1/n-th of the floors to check. For instance 1/3. Then the algorithm will look like the following:

Throw the egg from 33rd floor. If it breaks, then we check the first 32 floors using the second egg.

Otherwise, we throw the egg from 33 + (67 \* 1/3) = 55th floor. If it breaks, then we check floors 34 to 55 using the second egg.

Worst case scenario for 1/3 is max(33, 24, …) = 33. This way we might find a perfect n that optimizes the number of throws using some dynamic programming. This is a valuable solution that presents programming thinking, but it is not an optimal solution.

Your job is to write a function called bestWorstMax(e) that given a number of floors (e) finds the perfect split to find the minimum number of throws you need to find the floor at which the eggs are broken.

ESEMPIO CON DEI NUMERI…