4. Please Do Break the Crystal

Tell the broken plate you are sorry. – Mary Robertson.

Programming constructs and algorithmic paradigms covered in this puzzle: Break statements, radix representations.

You are tasked with determining the "hardness coefficient" of a set of *identical* crystal balls. The famous Shanghai Tower completed in 2015 has 128 floors, and you have to figure out from how high you can drop one of these balls so it doesn't break but rather bounces off the ground below. We will assume that the surrounding area has been evacuated while you conduct this important experiment.

What you have to report to your boss is the *highest* floor number of the Shanghai Tower from which the ball does *not* break when dropped. This means that if you report floor f, the ball does not break at floor f, but does break at floor $f + 1$. Else you would have reported $f + 1$. Your bonus depends on how high a floor you report, and if you report a floor f from which the ball breaks, you face a stiff fine, which you want to avoid at all costs.

Once a ball breaks, you can't reuse it again, but you can if it does not break. Since the ball's velocity as it hits the ground is the sole determining factor as to whether it breaks or not, and this velocity is proportional to the floor number, you can assume that if a ball does not break when dropped from floor x, it will not break from any floor whose number $< x$. Similarly, if it breaks when dropped from floor y, it will break when dropped from any floor whose number $> y$.

Sadly, you are not allowed to take an elevator because the shiny round objects you are carrying may scare off other passengers. You would therefore like to minimize the number of times you drop a ball, since it is a lot of work to keep climbing up stairs.

Of course, the big question is how many balls do you have to play with? Suppose you are given exactly one ball. You don't have much freedom to operate. If you drop the ball from floor 43, say, and it breaks, you don't dare report floor 42, because it might break when dropped from floor 42, floor 41, or floor 1 for that matter. You will have to report floor 1, which means no bonus. With one ball, you will have to start with floor 1 and if the ball breaks, report floor 1, and if it does not you move up to floor 2, all the way till floor 128. If it doesn't break at floor 128, you happily report 128. If the ball breaks when dropped from floor f, you will have dropped the ball f times. The number of drops could be as large as 128, floors 1 through 128 inclusive.

What if you have two balls? Suppose you drop one ball from floor 128. If it does not break, you report floor 128 and you are done with the experiment and rich. However, if it breaks, you are down to one ball and all you know is that the balls you are given definitely break at floor 128. To avoid a fine and to maximize your bonus, you will have start with the second ball at floor 1 and move up as described earlier possibly all the way up to floor 127. The number of drops in the worst case is 1 drop (from floor 128) plus drops from floors 1 through 127 inclusive, a total of 128. No improvement from the case of one ball ☹

Intuition says that you should guess the midpoint of the interval [1, 128] for the 128-floor building. Suppose you drop a ball at floor 64. There are two cases as always:

1. The ball breaks. This means that you can focus on floors 1 through 63, i.e., interval [1, 63] with the remaining ball.
2. The ball does not break. This means that you can focus on floors 65 through 128, i.e., interval [65, 128] with both balls.

The worst-case number of drops is 64 because in Case 1 you will need to start with the lowest floor in the interval and work your way up. Better than 128 but only by a factor of two.

You would like to do better than the worst case of 64 drops when you have two balls. You don't want to give up any part of your bonus, and a fine is a no-no.

Can you think of a way to maximize your bonus and avoid a fine while using no more than 21 drops in the case of two balls? What if you had more balls or what if the Shanghai Tower suddenly doubled in size in terms of number of floors?

Efficient Search with 2 Balls

We should be able to do better than 64 drops. The problem with beginning with floor 64 when we only have two balls is that if the first ball breaks, we have to start with floor 1 and go all the way to floor 63. What if we started at floor 20? If the first ball breaks, we have to search the smaller interval [1, 19] with the second ball one floor at a time. That is 20 drops total in the worst case. If the first ball does not break, we search the large interval [21, 128] but we have two balls. Let's next go to floor 40 and drop the first ball (second drop of the first ball). If the first ball breaks we search [21, 39] one floor at a time. This is, in the worst case, a total of 2 drops for the first ball (at floors 20 and 40) and 19 drops of the second ball for a total of 21. Onto floor 60 and so on. Trying floors 20, 40, 60, 80, etc., is going to get us a worst-case solution of less than 30 drops for sure.

Our purpose here is not to just solve a specific 128-floor problem. Is there a general algorithm where given an `n`-floor building and given two balls we can show a symbolic worst-case bound of `func(n)` where `func` is some function? Then, we can apply this algorithm to our specific 128-floor problem.

The key is to distribute the floors properly. Suppose we use a strategy where we drop the first ball at floors `k`, `2k`, `3k`, ..., `(n/k - 1)k, (n/k)k`. Let's assume that the first ball does not break till the last drop. In this case, we have `n/k` drops of the first ball, and have to search the interval `[(n/k - 1)k + 1, (n/k)k - 1]` which is `k - 1` drops of the second ball in the worst case. That is, a total of `n/k + k - 1` drops in the worst case.

Therefore, we want to choose `k` to minimize `n/k + k`. The minimum happens when `k` is $\sqrt{n}$.[1] The worst-case number of drops will be $2\sqrt{n} - 1$. For our 128-floor example, we should space the first ball's drops $\sqrt{128} = 11$ floors apart. This will get us to 21 drops worst-case ☺

Is 21 drops the best we can do? We did make the assumption that the floors we would drop from were evenly distributed, `k`, `2k`, `3k`, etc. It turns out that distributing the floors carefully and unevenly can shrink the number of drops required below 21. However, we will turn our focus to the general setting of having d >= 1 balls and n floors and see if we can come up with a strategy similar to the one for 1 ball and 2 balls, but which requires fewer drops as the number of balls increases.

Efficient Search with d Balls

When we have one ball, i.e., d = 1, we have no choice but to start at the first floor and work our way upwards. When we have two balls, we determined that starting at floor $\sqrt{n}$ (or $n^{½}$) is the way to go. Should we start at floor $n^{1/d}$ when we have d balls? What happens when a ball breaks?

[1] Occasionally, high school calculus comes in handy!

Let's consider r-ary representations of numbers. When $r = 2$, we have the binary representation, when $r = 3$, the ternary representation, and so on. Given the number of floors n, and the number of balls d, choose r such that $r^d > n$. So if $n = 128$, and $d = 2$, we will choose $r = 12$ as we did earlier. If $d = 3$, we will choose $r = 6$, since $5^3 < 128$ and $6^3 > 128$. Let's assume for the purposes of our next example that $d = 4$, which means $r = 4$, when $n = 128$.

The numbers we consider are d-digit, r-ary numbers. For our chosen $r = 4$, $d = 4$ example, the smallest number is 0000 (0 in decimal) and the largest number is 3333 (255 in decimal). As a reminder, in a radix-4 representation, a number such as 1233 is equivalent to $1 \times 4^3 + 2 \times 4^2 + 3 \times 4^1 + 3 \times 4^0 = 111$ in decimal.

We drop the first ball at floor 1000 (floor 64). If it does not break, we move to floor 2000 (floor 128) and drop the ball from there. If it does not break, we are done. If it does break, we are now down to 3 balls, but we know now that we only have to search the range [1001, 1333], which is [65, 127] in decimal. We will move to the second phase with the floor that corresponds to the highest floor in the first phase where the ball did not break.

In our second phase, we work with the second ball and the second digit from left of our r-ary representation. Let's assume that in the first phase the ball broke at floor 2000, and did not break at floor 1000. In the second phase, our first drop will be from floor 1100 (floor 80, which is inside the range [65, 127]). In the second phase we keep incrementing the second digit in our r-ary representation and drop balls from corresponding floors. We will drop from floor 1100, 1200 and 1300 in that order. As before, we move to the third phase with the floor that is the highest floor from which the ball did not break in the second phase. For the purposes of our example, let's assume that is floor 1200. We now need to search the interval [1201, 1233], since the ball broke at floor 1300. This corresponds to [97, 111] in decimal.

In Phase 3, we drop the third ball from floor 1210 – we increment the third digit of our representation from the floor we determined in the second phase. Floor 1210 is followed by 1220 and 1230. Let's assume the ball breaks when dropped from floor 1230. This means that we move to the fourth and final phase with floor 1220. The range we are searching in is [1221, 1223] corresponding to [105, 107] in decimal.

In our fourth and last phase, we drop the fourth ball from floors 1221, 1222 and 1223, incrementing the fourth digit in our representation. If the ball does not break, we report floor 1230. If the ball breaks when dropped from any of the floors, e.g., 1223, we report the floor that is one below, e.g., 1222.

What is the maximum number of drops that we will make? In each phase, we drop the ball $r - 1$ times at most. Since there are at most d phases, the total number of drops is at most $d \times (r - 1)$. For our example of $r = 4$, $d = 4$, we will have at most 12 drops with four balls on $n = 128$ floors! In fact, we will have at most 12 drops even for $n = 255$.

We need an interactive program embodying the algorithm above that will help us determine exactly the floors from which to drop balls from for arbitrary n and d, so we can efficiently determine the hardness coefficient of the given balls. This program will take as input n and d, and tell us from what floor to drop the first ball. Then, depending on the result – break or no break – that is input to the program, the program will either tell us a new floor to drop a (new) ball from, or tell us what the hardness coefficient is. The program terminates only when the hardness coefficient has been determined and it should tell us how many drops were needed in total.

Here's the code for our program:

```
 1.   def howHardIsTheCrystal(n, d):
 2.       r = 1
 3.       while (r**d <= n):
 4.           r = r + 1
 5.       print('Radix chosen is', r)
 6.       numDrops = 0
 7.       floorNoBreak = [0] * d
 8.       for i in range(d):
 9.           for j in range(r-1):
10.               floorNoBreak[i] += 1
11.               Floor = convertToDecimal(r, d, floorNoBreak)
12.               if Floor > n:
13.                   floorNoBreak[i] -= 1
14.                   break
15.               print ('Drop ball', i+1, 'from Floor', Floor)
16.               yes = input('Did the ball break (yes/no)?:')
17.               numDrops += 1
18.               if yes == 'yes':
19.                   floorNoBreak[i] -= 1
20.                   break
21.       hardness = convertToDecimal(r, d, floorNoBreak)
22.       return hardness, numDrops
```

Lines 2-5 determine the radix `r` that we need to use. We will use a list of numbers, each of which will vary from `0` to `r-1` as our representation for the floor. Our `d`-digit list representation `floorNoBreak` has the most significant digit to the left, i.e., index `0`, and is initialized to all `0`'s on Line 7. Line 8 begins the outer **`for`** loop corresponding to the `d` phases, and Line 9 begins the inner **`for`** loop corresponding to the drops of the chosen ball for the current phase.

We simply increment the appropriate digit in `floorNoBreak` corresponding to the phase we are in on Line 10. Given that `r**d` can be significantly larger than `n`, we need to check to make sure that we don't generate drops from floors larger than `n` – this is done in Lines 11-14. If the increment results in a floor larger than `n`, we are done with this phase and we immediately move to the next phase. That is what the **`break`** statement on Line 14 does; loop iterations end immediately, and the line immediately after the loop is executed. In this case it is Line 8, since we are breaking out of the inner **`for`** loop. Note that each **`break`** statement breaks out of the innermost loop that it is enclosed in; iterations in outer loops will continue. Line 11 invokes a simple function to convert from radix `r` to decimal that we will present later. If we are moving to the next phase, we need `floorNoBreak` to correspond to the highest floor from which an actual drop has not resulted in a break. This is why we decrement `floorNoBreak` on Line 13 before breaking out of the inner **`for`** loop.

We tell the user to drop a particular ball from a particular floor and wait for the result to be input by the user (Lines 15-16). If the ball does not break, we merely continue the loop. If the ball breaks, we need to set `floorNoBreak` to be the highest floor from which a drop has not resulted in a break, which requires a decrement as described above. We move to the next phase by breaking out of the inner **`for`** loop (Line 20).

Once we are done with all the phases, we have the hardness coefficient computed in `floorNoBreak` (Line 21).

The function `convertToDecimal` shown below takes an `r`-ary `d`-digit list representation and returns the decimal equivalent.

```
def convertToDecimal(r, d, rep):
    number = 0
    for i in range(d-1):
        number = (number + rep[i]) * r
    number += rep[d-1]
    return number
```

If we run the example we provided earlier:

```
howHardIsTheCrystal(128, 4)
```

We get the expected execution for the italicized yes/no user inputs below:

```
Radix chosen is 4
Drop ball 1 from Floor 64
Did the ball break (yes/no)?:no
Drop ball 1 from Floor 128
Did the ball break (yes/no)?:yes
Drop ball 2 from Floor 80
Did the ball break (yes/no)?:no
Drop ball 2 from Floor 96
Did the ball break (yes/no)?:no
Drop ball 2 from Floor 112
Did the ball break (yes/no)?:yes
Drop ball 3 from Floor 100
Did the ball break (yes/no)?:no
Drop ball 3 from Floor 104
Did the ball break (yes/no)?:no
```

```
Drop ball 3 from Floor 108
Did the ball break (yes/no)?:yes
Drop ball 4 from Floor 105
Did the ball break (yes/no)?:no
Drop ball 4 from Floor 106
Did the ball break (yes/no)?:no
Drop ball 4 from Floor 107
Did the ball break (yes/no)?:yes
```

The program returns a hardness coefficient of `106`, and a number of drops of `11`.

Reducing The Number of Drops for 2 Balls

Our algorithm drops balls from that the floors that are evenly distributed, `k`, `2k`, `3k`, etc. Let's look at what was lost because of this assumption. For n = 100, and with 2 balls with hardness coefficient of 65, our algorithm makes drops of the first ball from floors 11, 22, 33, 44, 55 and 66. If the first ball breaks when dropped from floor 66, the algorithm drops the second ball from floor 56, then 57, all the way to floor 65. When the ball does not break at floor 65, it reports the hardness coefficient of 65. This requires 16 drops in total.

Distributing the floors carefully and unevenly can shrink the number of drops required to a *maximum* of 14 for a 100-floor building. Let's say we want a maximum of k drops. For our *first* drop if we drop the (first) ball from the kth floor, and it breaks, we require at most k – 1 drops of the second ball to find out which floor the balls break. If the first ball does not break when dropped from the kth floor, we next drop this ball from the k + (k – 1)th floor. Why? If the ball breaks, we require k – 2 drops to discover the lowest floor from which the ball breaks. This would be a total of k drops, 2 for the first ball and k – 2 for the second ball.

Going on like this, we can relate k to the number of floors in the building, n. We can write:

$$n \leq k + (k - 1) + (k - 2) + (k - 3) \ldots + 2 + 1$$

This means that $n \leq k(k + 1)/2$. For n = 100, k = 14. We should drop the balls, from floors 14, 27, 39, 50, 60, 69, 77, 84, 90, 95, 99 and 100. For example, if the first ball breaks from floor 99 on the 11th drop, we drop the second ball from floor 96, 97 and 98 in the worst case.

Exercises

Exercise 1: If you run `howHardIsTheCrystal(128, 6)`, you see:

```
Radix chosen is 3
Drop ball 2 from Floor 81
```

The first drop is done using ball 2. What has happened here is that $2^6 < 128$, and so $r = 3$ is chosen. But not only is $3^6 > 128$, so is $3^5 = 243$. Our algorithm skips over the first ball, since the first digit in our representation is always 0. Fix the code so it removes unnecessary balls and informs the user about the number of balls it is actually using. Your modified program should always start with dropping ball 1 from some floor.

Exercise 2: Modify the code so it prints the number of balls that were used, i.e., broken.

Exercise 3: Modify the code to print out the interval of floors currently under consideration. Initially, the interval is `[0, n]`. This interval keeps shrinking as ball drop results are entered into the program. Your code should print the new interval each time a result is entered by the user. The hardness coefficient corresponds to the final interval with a size of one floor.