The tower and marbles puzzle

Eric Martin, CSE, UNSW

COMP9021 Principles of Programming, trimester 1, 2019

```
In [1]: from math import sqrt, ceil
        from random import randint
```

In order to test the quality of the products of a marble manufacturer, one plans to drop marbles from various levels of a tower and see whether they break. We make the following assumptions:

- If a marble breaks when dropped from a given level, then it would have broken if dropped from a higher level.
- All marbles are of the same quality: they all break or all do not break if dropped from a given level.

One intends to find out the highest level L such that a marble dropped from level L does not break, but breaks if dropped from a higher level, if any. The aim is to minimise the effort, that is:

- find out the maximum number of drops d needed in the worst case to get the correct answer;
- come up with a strategy to, for any marble quality, get the correct answer with no more than d drops.

The problem has two parameters:

- the number of available marbles, m;
- the number of levels, n.

如果m等于1，解决方案很简单。然后d等于n，唯一有效的策略包括从级别1，然后从级别2，然后从级别3 ..... 直到它中断，如果它曾经。 如果大理石质量最好并且从任何水平下降时不会破裂，或者如果它们具有次佳质量并且从下降时掉落则需要最多n滴。最高级别，但不是从下面的任何级别。
对于m的任何值都存在一般解，但是在m等于2的情况下存在更简单的解；在描述和实现前者之前，我们描述并实现后者。 所以现在，我们将m的值设置为2

Both m and n are assumed to be nonnegative integers.

The solution is straightforward if m is equal to 1. Then d is equal to n and the only valid strategy consists in dropping the unique marble from level 1, then from level 2, then from level 3... until it breaks, if it ever does. The maximum of n drops is needed if the marbles are of the best quality and do not break when dropped from any level, or if they are of the next best quality and break when dropped from the highest level, but not from any level below.

There exists a general solution for any value of m, but a simpler solution exists in case m is equal to 2; before describing and implementing the former, we describe and implement the latter. So for now, we set the value of m to 2.

我们不是直接从n计算d，而是反过来解决问题：给定d，最大级别数n（塔的最大高度）是多少，这样对于任何大理石质量，人们肯定会
正确评估弹珠的质量不超过d滴？如果第一块大理石破碎，从L层下降，只留下第二块大理石，没有其他选择，但探索L以下的所有层面，如果有的话，从最低层开始，并在未探测到的所有层面上 水平低于L.

Rather than directly computing d from n, we reverse the problem: given d, what is the maximum number of levels n (the maximum height of the tower) such that for any marble quality, one is sure to correctly assess the marbles' quality with no more than d drops? If the first marble ever breaks, dropped from level L, one is left with the second marble only and has no other option but explore all levels below L, if any, starting with the lowest such level, and making one's way up on all unexplored levels below L.

- This means that the first drop should be from level d: if the marble breaks, then there are $d-1$ drops left and the second marble has to be dropped from level 1, then from level 2... potentially up to level $d-1$ (in case the lowest level from which marbles break is either level d or level $d-1$). It is best to make the first drop from as high as possible, so the first drop should be made from level d.

这意味着第一掉落应该来自d级：如果大理石断裂，那么剩下d-1掉落，第二块大理石必须从1级掉落，然后从2级掉落.....可能达到水平 d - 1（如果大理石破裂的最低等级是等级d或等级d - 1）。 最好尽可能高地进行第一次下降，因此第一次下降应该从d级开始

同样的推理表明，如果第一个大理石在第一次跌落后没有破裂，那么它应该从d + (d - 1) 水平下降：如果它破裂，那么还有d - 2掉落并且有d -2 水平d之间有2个未探索水平，已知从L下降的大理石不会破裂的最高水平L，以及水平d + (d-1)，其中已知大理石从大理石下降的最低水平L0 L0确实破了。
第二个大理石应该从d + 1级下降，然后从d + 2级下降.....可能上升到2d-2级（如果大理石中断的最低级别是2d-1级或2d-2级）

- The same reasoning shows that in case the first marble does not break following the first drop, it should then be dropped from level $d + (d - 1)$: if it breaks, then there are $d - 2$ drops left and there are $d - 2$ unexplored levels between level d, the highest level L for which it is known that a marble dropped from L does not break, and level $d + (d - 1)$, the lowest level L' for which it is known that a marble dropped from L' does break. The second marble should be dropped from level $d + 1$, then from level $d + 2$... potentially up to level $2d - 2$ (in case the lowest level from which marbles break is either level $2d - 1$ or level $2d - 2$). 如果跟随第二次下降，第一次大理石不会破裂，第三次下降应该来自等级d + (d - 1) + (d - 2)
- If following the second drop, the first marble does not break, the third drop should be from level $d + (d - 1) + (d - 2)$. 如果大理石质量最好并且永不破裂，那么第一块大理石将被使用，从等级d下降，然后从等级d + (d-1) 下降，然后从等级d + (d-1) + (d- 2) ...并且最终，对于第d次跌落，从等级d + (d-1) + (d-2) + · · · + 2 + 1
- ...
- In case marbles are of the best quality and never break, the first marble only will be used, dropped from level d, then from level $d + (d - 1)$, then from level $d + (d - 1) + (d - 2)$... and eventually, for the dth drop, from level $d + (d - 1) + (d - 2) + \cdots + 2 + 1$.

This shows that n is equal to $\frac{d(d+1)}{2}$; n and d satisfy the equality $d^2 + d - 2n = 0$.

Of course, only some nonnegative integers n are of the form $\frac{d(d+1)}{2}$ for some nonnegative integers d. In the original problem, n is known and arbitrary, while d has to be computed from n. We infer from the previous considerations that d is the smallest nonnegative integer such that $d^2 + d - 2n \geq 0$, hence $d = \lceil \frac{-1+\sqrt{1+8n}}{2} \rceil$.

- Assume that $n = 6$; then $d = 3$. The first drop should be from level 3, and if the marble does not break, the second drop should be from level $3 + 2 = 5$, and if the marble still does not break, the third drop should be from level $3 + 2 + 1 = 6$.
- Assume that $n = 5$; then $d = 3$ too. The first drop should be from level 3, and if the marble does not break, the second drop should be from level $3 + 2 = 5$, and if the marble does not break, we are done, as there is no higher level to explore.
- Assume that $n = 4$; then $d = 3$ still. The first drop should be from level 3. If the marble does not break, the second drop could afford to be from level $3 + 2 = 5$, but since there is no such level, it should be from level 4.

This illustrates that given i between 1 and d, there can only be an ith drop with the first marble in case n is at least equal to $1 + \Sigma_{j=0}^{i-2}(d - j)$, and if there is an ith drop with the first marble, then it should be from level $\min(n, \Sigma_{j=0}^{i-1}(d - j))$.

变量 We define a variable, `breaking_level`, meant to denote the value of the lowest level L such that a marble dropped from L breaks, if there is such a level. We should also allow for the case where the marble is of best quality and does not break when dropped from any level. To simplify the design of the solution, it is best to let $n + 1$ be the corresponding value. One can think of $n + 1$ to be an additional, "virtual" level, such that any marble dropped from that level is sure to break, which is consistent with the reasonable assumption that a marble is sure to break when dropped from high enough.

我们定义一个变量breaking_level，意思是表示最低等级L的值，这样如果存在这样的等级，从L处掉落的大理石就会断裂。 我们也应该考虑大理石的情况质量最好，从任何级别下降都不会破坏。 为了简化解决方案的设计，最好让n + 1为相应的值。 人们可以认为n + 1是一个额外的“虚拟”水平，这样从那个水平下降的任何大理石肯定会破裂，这与合理的假设是一致的，即大理石在从足够高处掉落时肯定会断裂。

We define two variables, `low` and `high`, meant to keep track of the highest level L for which it is known that a marble does not break when dropped from level L, and of the lowest level L' for which it is known that a marble does break when dropped from level L', respectively. The initialisation should capture what we know before the procedure starts: `low` should be set to `0` and `high` to $n + 1$. At any stage of the computation, `low` will be smaller than `high`.

There is some uncertainly for as long as there is a gap between `low` and `high`, so for as long as `low` is strictly smaller than `high - 1`. The next drop will then be from level l for some l strictly between `low` and `high`, raising the value of `low` to l in case the marble does not break, decreasing the value of `high` to l in case the marble breaks, in both cases, reducing the gap between `low` and `high`.

我们定义了两个变量，低和高，用于跟踪最高级别L，已知大理石在从L级落下时不会断裂，以及最低级别L0，已知大理石确实 分别从L0级别下降时中断。 初始化应该在过程开始之前捕获我们所知道的：低应该设置为0并且高到n + 1.在计算的任何阶段，低将小于高。
只要低和高之间存在差距，就会有一些不确定因素，因此只要低点严格小于高点-1，那么下一个点将从低位l开始，对于某些低点严格地介于低点之间在大理石不破裂的情况下，将低值提高到l，在大理石破碎的情况下将高值降低到l，在这两种情况下，减小低和高之间的差距

当使用第一个大理石时，第一个下降应该是当前值低于d的水平，设置为0，第二个下降应该是低于当前值的d - 1水平，在第一个下降后变为d，第三个下降 应该是低于当前低值的d - 2级，在第二次降低后改为d + (d - 1) .....在每种情况下除非计算值至少等于高，否则应该
然后设置为高 - 1。
• 使用第二块大理石时，每次跌落应低于当前低值的1级。

- When using the first marble, the first drop should be d levels above the current value of `low`, set to `0`, the second drop should be $d-1$ levels above the current value of `low`, changed to d after the first drop, the third drop should be $d-2$ levels above the current value of `low`, changed to $d+(d-1)$ after the second drop... in each case unless the computed value is at least equal to `high`, as it should then be set to `high - 1`.
- When using the second marble, every drop should be 1 level above the current value of `low`.

变量步骤可以跟踪当前低于当前值的多少级别以进行下一次下降

A variable `step` can keep track of by how many levels above the current value of `low` to go up for the next drop.

After the last drop, `high` is equal to `low + 1`, and it is known that:

- if `high` is equal to $n+1$, then marbles are of best quality and do not break;
- if `high` is strictly smaller than $n+1$, then a marble breaks when dropped from level `high`, but not below.

意见 系统

The following function puts together all observations that precede, and allows one to systematically test the strategy, in all cases, for a given value of n:

```
In [2]: def systematic_two_marbles_simulation(n):
            d = ceil((sqrt(8 * n + 1) - 1) / 2)
            if d == 0:
                print('No drop will be needed.')
            elif d == 1:
                print('At most 1 drop will be needed.')
            else:
                print('At most', d, 'drops will be needed.')
            for breaking_level in range(1, n + 2):
                if breaking_level <= n:
                    print('\nStrategy for marbles breaking on level '
                          f'{breaking_level} and not below:'
                         )
                else:
                    print('\nStrategy for marbles not breaking:')
                low = 0
                high = n + 1
                step = d
                while low < high - 1:
                    level = min(low + step, high - 1)
                    if breaking_level <= level:
                        print(f'From level {level}... marble breaks!')
                        high = level
                        step = 1
                    else:
                        print(f'From level {level}... marble does not break!')
                        low = level
                        # If we are using the second marble,
                        # step is equal to 1 and should remain equal to 1.
                        if step > 1:
                            step -= 1
```

```
In [3]: systematic_two_marbles_simulation(1)

At most 1 drop will be needed.
```
战略，策略
```
Strategy for marbles breaking on level 1 and not below:
From level 1… marble breaks!

Strategy for marbles not breaking:
From level 1… marble does not break!


In [4]: systematic_two_marbles_simulation(2)

At most 2 drops will be needed.

Strategy for marbles breaking on level 1 and not below:
From level 2… marble breaks!
From level 1… marble breaks!

Strategy for marbles breaking on level 2 and not below:
From level 2… marble breaks!
From level 1… marble does not break!

Strategy for marbles not breaking:
From level 2… marble does not break!


In [5]: systematic_two_marbles_simulation(3)

At most 2 drops will be needed.

Strategy for marbles breaking on level 1 and not below:
From level 2… marble breaks!
From level 1… marble breaks!

Strategy for marbles breaking on level 2 and not below:
From level 2… marble breaks!
From level 1… marble does not break!

Strategy for marbles breaking on level 3 and not below:
From level 2… marble does not break!
From level 3… marble breaks!

Strategy for marbles not breaking:
From level 2… marble does not break!
From level 3… marble does not break!


In [6]: systematic_two_marbles_simulation(4)
```

```
At most 3 drops will be needed.

Strategy for marbles breaking on level 1 and not below:
From level 3… marble breaks!
From level 1… marble breaks!

Strategy for marbles breaking on level 2 and not below:
From level 3… marble breaks!
From level 1… marble does not break!
From level 2… marble breaks!

Strategy for marbles breaking on level 3 and not below:
From level 3… marble breaks!
From level 1… marble does not break!
From level 2… marble does not break!

Strategy for marbles breaking on level 4 and not below:
From level 3… marble does not break!
From level 4… marble breaks!

Strategy for marbles not breaking:
From level 3… marble does not break!
From level 4… marble does not break!
```

```
In [7]: systematic_two_marbles_simulation(5)

At most 3 drops will be needed.

Strategy for marbles breaking on level 1 and not below:
From level 3… marble breaks!
From level 1… marble breaks!

Strategy for marbles breaking on level 2 and not below:
From level 3… marble breaks!
From level 1… marble does not break!
From level 2… marble breaks!

Strategy for marbles breaking on level 3 and not below:
From level 3… marble breaks!
From level 1… marble does not break!
From level 2… marble does not break!

Strategy for marbles breaking on level 4 and not below:
From level 3… marble does not break!
From level 5… marble breaks!
From level 4… marble breaks!
```

```
Strategy for marbles breaking on level 5 and not below:
From level 3… marble does not break!
From level 5… marble breaks!
From level 4… marble does not break!

Strategy for marbles not breaking:
From level 3… marble does not break!
From level 5… marble does not break!
```

```
In [8]: systematic_two_marbles_simulation(6)
```

```
At most 3 drops will be needed.

Strategy for marbles breaking on level 1 and not below:
From level 3… marble breaks!
From level 1… marble breaks!

Strategy for marbles breaking on level 2 and not below:
From level 3… marble breaks!
From level 1… marble does not break!
From level 2… marble breaks!

Strategy for marbles breaking on level 3 and not below:
From level 3… marble breaks!
From level 1… marble does not break!
From level 2… marble does not break!

Strategy for marbles breaking on level 4 and not below:
From level 3… marble does not break!
From level 5… marble breaks!
From level 4… marble breaks!

Strategy for marbles breaking on level 5 and not below:
From level 3… marble does not break!
From level 5… marble breaks!
From level 4… marble does not break!

Strategy for marbles breaking on level 6 and not below:
From level 3… marble does not break!
From level 5… marble does not break!
From level 6… marble breaks!

Strategy for marbles not breaking:
From level 3… marble does not break!
From level 5… marble does not break!
From level 6… marble does not break!
```

为了模拟策略，最好不要修复breaking_level的值，而是随机生成1到n + 1之间的数字，这将保持“ idden ”，并且模拟策略将显示出来。为此目的， 我们用随机模块中的randint () 函数。 与randrange () 相反，randrange () 与range () 一样，排除了作为第二个参数提供的上限，randint () 确实包含上限。 下面说明，同时还演示了如何从dict类中的fromkeys () 方法允许从一个iterable创建一个字典来生成键，所有值都设置了默认情况下为none，可以通过为fromkeys () 提供第二个参数来更改，此处为0：

模拟

To simulate the strategy, it is best not to fix the value of `breaking_level`, but instead randomly generate a number between 1 and $n+1$, that will remain "hidden" and that the simulated strategy will reveal. For this purpose, we use the `randint()` function from the `random` module. In contrast to `randrange()` which, like `range()`, excludes the upper bound provided as second argument, `randint()` does include the upper bound. The following illustrates, while also demonstrating how the `fromkeys()` method from the `dict` class allows one to create a dictionary from an iterable to generate the keys, with values all set to `None` by default, which can be changed by providing a second argument to `fromkeys()`, here, `0`:

```
In [9]: random_numbers = dict.fromkeys(range(1, 5), 0)
        for _ in range(1000):
            random_numbers[randint(1, 4)] += 1

        random_numbers

Out[9]: {1: 237, 2: 282, 3: 263, 4: 218}
```

模拟

The function defined next is hardly different to `systematic_two_marbles_simulation()`. Besides assigning `breaking_level` a random value, it does not make use of `step` as `d` suffices and can be modified to play the role that `step` plays in `systematic_two_marbles_simulation()`. The output is made more precise, keeping track of the drop numbers, and which one of the first or second marble is used:

接下来定义的函数与syste_two_marbles_simulation () 几乎有什么不同。 除了为break_level分配一个随机值之外，它不会使用step作为d就足够了，并且可以修改它以扮演step在sys e_two_marbles_simulation () 中扮演的角色。 输出更加精确，跟踪掉落数量，以及使用第一个或第二个大理石中的哪一个：

```
In [10]: def two_marbles_simulation(n):
             print(f'Experimenting with {n} levels.'
             d = ceil((sqrt(8 * n + 1) - 1) / 2)
             if d == 0:
                 print('No drop will be needed.')
             elif d == 1:
                 print('At most 1 drop will be needed.\n')
             else:
                 print('At most', d, 'drops will be needed.\n')
             print('Now simulating...\n')
             low = 0
             high = n + 1
             drop = 0
             which_marble = 'first'
             breaking_level = randint(1, n + 1)
             while low < high - 1:
                 level = min(low + d, high - 1)
                 drop += 1
                 if breaking_level <= level:
                     print(f'Drop #{drop} with {which_marble} marble, '
                           f'from level {level}... marble breaks!'
                          )
                     which_marble = 'second'
                     high = level
                     d = 1
                 else:
                     print(f'Drop #{drop} with {which_marble} marble, '
                           f'from level {level}... marble does not break!'
```

```
                    )
                low = level
                if d > 1:
                    d -= 1
        if high == n + 1:
            print("Tower would have to be higher to reveal marbles's limits.")
        elif high == 1:
            print(f'Marbles break when dropped from the first level.')
        else:
            print(f'Marbles break when dropped from level {high}, not below.')
```

```
In [11]: two_marbles_simulation(30)

Experimenting with 30 levels.
At most 8 drops will be needed.

Now simulating…

Drop #1 with first marble, from level 8… marble does not break!
Drop #2 with first marble, from level 15… marble does not break!
Drop #3 with first marble, from level 21… marble does not break!
Drop #4 with first marble, from level 26… marble breaks!
Drop #5 with second marble, from level 22… marble does not break!
Drop #6 with second marble, from level 23… marble breaks!
Marbles break when dropped from level 23, not below.
```

```
In [12]: two_marbles_simulation(0)

Experimenting with 0 levels.
No drop will be needed.
Now simulating…

Tower would have to be higher to reveal marbles's limits.
```

让我们设计并实现任意m值的解决方案。 我们仍然可以解决这个问题：给定d，最大等级数n（塔的最大高度）是多少，这样对于任何大理石质量，人们都可以正确地评估大理石的质量而不超过d滴？ 现在，n取决于d和m；写n（d，m）为n。 平凡地，如果d = 0或m = 0，则H（d，m）= 0。假设d和m都是严格正的

Let us design and implement the solution for an arbitrary value of m. We still reverse the problem: given d, what is the maximum number of levels n (the maximum height of the tower) such that for any marble quality, one is sure to correctly assess the marbles' quality with no more than d drops? Now, n depends on both d and m; write $H(d, m)$ for n. Trivially, if either $d = 0$ or $m = 0$, then $H(d, m) = 0$. Assume that both d and m are strictly positive.

如果大理石断裂，剩下m - 1弹珠，不需要在更高水平上进行测试。

- If the marble breaks, m - 1 marbles remain and no test needs to be done on higher levels.
- If the marble does not break, m marbles remain and no test needs to be done on lower levels.
- In any case, d - 1 drops remain.

• 如果大理石没有破裂，则会留下大理石，并且不需要在较低的水平上进行测试。

This yields: $H(d, m) = H(d - 1, m - 1) + H(d - 1, m) + 1$.

These equations allow one to compute d as the least integer with $H(d, m) >= n$. For the simulation, if *low* is the largest integer for which it is known that marbles can be dropped from level *low* without

• 无论如何，d - 1保持不变。

这些等式允许人们将d计算为H（d，m）> = n的最小整数。 对于模拟，如果低是最大的整数，已知大理石可以从低水平下降而不破裂，d是剩余的水滴数量，并且错过了剩余的大理石数量，那么下一个下降应该来自 低水平+ H（d0 - 1，m0 - 1）+ 1，剩余的水滴和理石的数量将根据后者的最后一次下降的结果进行简单更新，但前者不会更新。

breaking, d' is the number of drops that remain, and m' is the number of marbles that remain, then the next drop should be from level $low + H(d' - 1, m' - 1) + 1$, and the numbers of drops and marbles that remain will be trivially updated, depending on the outcome of the last drop for the latter, but not for the former.

For all nonnegative integers k and n, set

- $B(0, k) = 1$,
- $B(n, 0) = 1$, and
- $B(n + 1, k + 1) = B(n, k) + B(n, k + 1)$.

这些递归关系与确定二项式系数的那些相同。 让我们验证以下内容

These recurrence relations are identical to those that determine the binomial coefficients. Let us verify the following:

1. $H(n, k) = B(n, k) - 1$.
2. If $k > n$ then $B(n, k) = B(n, n)$.
3. If $k <= n$ then $B(n, k) = \Sigma_{k'=0}^{k} \binom{n}{k'}$.

For 1.: The equality is trivial if either $n = 0$ or $k = 0$. Suppose that both $n > 0$ and $k > 0$, and $H(n', k') = B(n', k') - 1$ for all $n' \leq n$ and $k' \leq k$ with either $n' < n$ or $k' < k$. Then $H(n, k)$ is equal to $H(n–1, k–1) + H(n–1, k) + 1$, which is equal to $B(n–1, k–1) - 1 + B(n–1, k) - 1 + 1$, which is equal to $B(n–1, k–1) + B(n–1, k) - 1$, which is equal to $B(n, k) - 1$.

For 2.: The equality is trivial if $n = 0$. Suppose that $n > 0$ and that $B(n', k) = B(n', n')$ for all $n' < n$ and $k > n'$. Then for all $k > n$, $B(n, k)$ is equal to $B(n - 1, k - 1) + B(n - 1, k)$, which is equal to $B(n - 1, n - 1) + B(n - 1, n)$, which is equal to $B(n, n)$.

For 3.: The equality is trivial if $k = 0$. Suppose that $k > 0$, and that the equality holds for any $k' < k$ and $n' \geq k'$. Then for all $n \geq k$, $B(n, k) = B(n - 1, k - 1) + B(n - 1, k) = \Sigma_{k'=0}^{k-1} \binom{n-1}{k'} + \Sigma_{k'=0}^{k} \binom{n-1}{k'} = \Sigma_{k'=1}^{k} [\binom{n-1}{k'-1} + \binom{n-1}{k'}] + \binom{n-1}{0} = \Sigma_{k'=1}^{k} \binom{n}{k'} + \binom{n}{0} = \Sigma_{k'=0}^{k} \binom{n}{k'}$.

The values $B(n, k)$ determine Bernouilli rectangle, whose rows are computed similarly to the rows of Pascal triangle:

值B（n，k）确定伯努利矩形，其行的计算方式与Pascal三角形的行类似

1	1	1	1	1	1	1	...
1	2	2	2	2	2	2	...
1	3	4	4	4	4	4	...
1	4	7	8	8	8	8	...
1	5	11	15	16	16	16	...
1	6	16	26	31	32	32	...
1	7	22	42	57	63	64	...
...	...	...	...	...	...	...	...

配料

让我们总结一下关键成分，感谢d可以计算出

Let us summarise the key ingredients thanks to which d can be computed:

- $H(d, m) = B(d, m) - 1$
- $B(d, m)$ is equal to the value of Bernouilli triangle at the intersection of row number d and column number m (with both numberings starting from 0).
- $H(d, m)$ is the maximum number of levels a tower can have so that with at most d drops and using at most m marbles, it is possible to know the lowest L, if any, such that a marble dropped from L breaks but does not break when dropped from below L.
- m, the number of marbles, and n, the tower height, are known, and d has to be determined from m and n.

It follows that:

- d is the least integer such that $H(m, d) \geq n$, hence the least integer such that $B(m, d) > n$;
- to compute d, it suffices to build the first m columns of Bernouilli triangle for row 1, row 2... up to the first row whose mth column contains a number greater than n. For instance, if $m = 4$ and $n = 25$, then $d = 5$ since column number 4 of Bernouilli rectangle reads as 1, 2, 4, 8, 16, 31..., with 31, first number in the sequence greater than 25, on row number 5.

非负整数

Given nonnegative integers p and q, the first p rows of the first q columns of Bernouilli rectangle can be represented as a list of p lists, each of which is a list of q numbers. For instance, the first 3 rows of the first 4 columns of Bernouilli rectangle can be represented as:

可以表示伯努利矩形的前4列的前3行

```
In [13]: [[1, 1, 1, 1],
          [1, 2, 2, 2],
          [1, 3, 4, 4]
         ]
```

```
Out[13]: [[1, 1, 1, 1], [1, 2, 2, 2], [1, 3, 4, 4]]
```

Before we build the section of Bernouilli rectangle of interest thanks to which we can determine d, and then run the simulation, let us work with a rectangle of size 8 x 10, filled with all integers from 0 up to 79, the integers between 0 and 9 making up the first row, those between 10 and 19 the second row, etc. To this end, we create a list `rectangle` of 8 elements, each of which is a list of 10 elements; each element of `rectangle` represents a row and is conveniently defined with a list comprehension, while `rectangle` itself represents the sequence of all rows and is also conveniently defined with a list comprehension:

下划线

理解

```
In [14]: rectangle = [[10 * i + j for j in range(10)] for i in range(8)]

         rectangle
```

```
Out[14]: [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
          [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
          [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
          [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
          [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
          [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
          [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
          [70, 71, 72, 73, 74, 75, 76, 77, 78, 79]]
```

Let us write a function to display the contents of `rectangle` in a way that is consistent with that representation. The largest number, 79, is stored in the bottom right corner: it is the last member of the last member of `rectangle`, hence it can be retrieved as `rectangle[-1][-1]`. It consists of 2 digits, with 2 equal to `len(str(79))`. The `display()` function is appropriate to display the contents of any rectangle of integers by nicely aligning all values in a given column, under the assumption that the space taken by integers in the rectangle is maximal for the integer in the bottom right corner:

```
In [15]: def display(rectangle):
             field_width = len(str(rectangle[-1][-1]))
             for row in rectangle:
                 print(' '.join(f'{e:{field_width}}' for e in row))

         display(rectangle)
```

```
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
```

When defining `rectangle`, we could have started with an empty list L, and 8 times, appended L with an appropriate list of 10 elements:

```
In [16]: rectangle = []
         for i in range(8):
             rectangle.append([10 * i + j for j in range(10)])

         display(rectangle)
```

```
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
```

我们使用这种技术来构建感兴趣的伯努利矩形部分，从列表中列出一个列表而不是空列表。 我们逐行追踪建筑。 最终的行数
构造比最大滴数多1

We use this technique to build the section of Bernouilli rectangle of interest, starting with a list of one list rather than with an empty list. We trace the construction row by row. The number of rows eventually constructed is 1 more than the maximum of drops:

```
In [17]: def bernouilli_rectangle(m, n):
             bernouilli_rows = [[1] * (m + 1)]
             while bernouilli_rows[-1][m] <= n:
                 row = bernouilli_rows[-1]
                 bernouilli_rows.append([1] + [row[i - 1] + row[i]
                                               for i in range(1, len(row))
                                              ]
                                        )
                 display(bernouilli_rows)
                 print()
             return bernouilli_rows

         bernouilli_rows = bernouilli_rectangle(4, 25)
         print('The maximum number of drops is', len(bernouilli_rows) - 1)
```

```
1 1 1 1 1
1 2 2 2 2
```

```
1 1 1 1 1
1 2 2 2 2
1 3 4 4 4

1 1 1 1 1
1 2 2 2 2
1 3 4 4 4
1 4 7 8 8

 1  1  1  1  1
 1  2  2  2  2
 1  3  4  4  4
 1  4  7  8  8
 1  5 11 15 16

 1  1  1  1  1
 1  2  2  2  2
 1  3  4  4  4
 1  4  7  8  8
 1  5 11 15 16
 1  6 16 26 31

The maximum number of drops is 5
```

模拟非常类似于2个弹珠的情况。 关键的区别在于计算大理石应该从哪个级别下降：第一个d减1，然后赋值级别= min（低+ d，高 - 1）变为level = min（低+ bernoulli_rows） [d] [m - 1]，高 - 1）。

模拟

The simulation is very similar to the 2 marbles case. The key difference is in computing the level from which a marble should then be dropped: first `d` is decremented by 1, and then the assignment `level = min(low + d, high - 1)` is changed to `level = min(low + bernouilli_rows[d][m - 1], high - 1)`. Indeed:

- we noticed that if *low* is the largest integer for which it is known that marbles can be dropped from level *low* without breaking, d' is the number of drops that remain, and m' is the number of marbles that remain, then the next drop should be from level $low + H(d' - 1, m' - 1) + 1$;
- we established that $H(d', m') = B(d', m') - 1$.

The necessary values (and others) and precomputed in `bernouilli_rows`, which allows the rest of the code to be slightly simpler than with the 2 marbles case:

```
In [18]: def m_marbles_simulation(m, n):
             print(f'Experimenting with {m} marbles and {n} levels.')
             bernouilli_rows = [[1] * (m + 1)]
             while bernouilli_rows[-1][m] <= n:
                 row = bernouilli_rows[-1]
                 bernouilli_rows.append([1] + [row[i - 1] + row[i]
                                               for i in range(1, len(row))
                                              ]
                                        )
             d = len(bernouilli_rows) - 1
```

```
    if d == 0:
        print('No drop will be needed.')
    elif d == 1:
        print('At most 1 drop will be needed.\n')
    else:
        print('At most', d, 'drops will be needed.\n')
    print('Now simulating...\n')
    low = 0
    high = n + 1
    drop = 0
    which_marble = 1
    breaking_level = randint(1, n + 1)
    while low < high - 1:
        d -= 1
        level = min(low + bernouilli_rows[d][m - 1], high - 1)
        drop += 1
        if breaking_level <= level:
            print(f'Drop #{drop} with marble #{which_marble}, '
                  f'from level {level}... marble breaks!'
                 )
            which_marble += 1
            high = level
            m -= 1
        else:
            print(f'Drop #{drop} with marble #{which_marble}, '
                  f'from level {level}... marble does not break!'
                 )
            low = level
    if high == n + 1:
        print("Tower would have to be higher to reveal marbles's limits.")
    elif high == 1:
        print(f'Marbles break when dropped from the first level.')
    else:
        print(f'Marbles break when dropped from level {high}, not below.')
```

```
In [19]: m_marbles_simulation(4, 20)
```

```
Experimenting with 4 marbles and 20 levels.
At most 5 drops will be needed.

Now simulating…

Drop #1 with marble #1, from level 15… marble breaks!
Drop #2 with marble #2, from level 7… marble does not break!
Drop #3 with marble #2, from level 11… marble breaks!
Drop #4 with marble #3, from level 9… marble breaks!
Drop #5 with marble #4, from level 8… marble breaks!
Marbles break when dropped from level 8, not below.
```

```
In [20]: m_marbles_simulation(7, 2000)

Experimenting with 7 marbles and 2000 levels.
At most 12 drops will be needed.

Now simulating…

Drop #1 with marble #1, from level 1486… marble breaks!
Drop #2 with marble #2, from level 638… marble does not break!
Drop #3 with marble #2, from level 1020… marble does not break!
Drop #4 with marble #2, from level 1239… marble breaks!
Drop #5 with marble #3, from level 1119… marble breaks!
Drop #6 with marble #4, from level 1062… marble does not break!
Drop #7 with marble #4, from level 1088… marble does not break!
Drop #8 with marble #4, from level 1103… marble does not break!
Drop #9 with marble #4, from level 1111… marble does not break!
Drop #10 with marble #4, from level 1115… marble breaks!
Drop #11 with marble #5, from level 1113… marble does not break!
Drop #12 with marble #5, from level 1114… marble breaks!
Marbles break when dropped from level 1114, not below.
```