**Egg Drop**

**Problem**

Imagine that you are in a building with F floors (starting at floor 1, the lowest floor), and you have a large number of identical eggs, each in its own identical protective container. For each floor in the building, you want to know whether or not an egg dropped from that floor will break. If an egg breaks when dropped from floor i, then all eggs are guaranteed to break when dropped from any floor j ≥ i. Likewise, if an egg doesn't break when dropped from floor i, then all eggs are guaranteed to never break when dropped from any floor j ≤ i.

We can define Solvable(F, D, B) to be true if and only if there exists an algorithm to determine whether or not an egg will break when dropped from any floor of a building with F floors, with the following restrictions: you may drop a maximum of D eggs (one at a time, from any floors of your choosing), and you may break a maximum of B eggs. You can assume you have at least D eggs in your possession.

**Input**

The first line of input gives the number of cases, N. N test cases follow. Each case is a line formatted as:

F D B

Solvable(F, D, B) is guaranteed to be true for all input cases.

**Output**

For each test case, output one line containing "X: " followed by three space-separated integers: Fmax, Dmin, and Bmin. The definitions are as follows:

Fmax is defined as the largest value of F' such that Solvable(F', D, B) is true, or -1 if this value would be greater than or equal to 232 (4294967296).

(In other words, Fmax = -1 if and only if Solvable(232, D, B) is true.)

Dmin is defined as the smallest value of D' such that Solvable(F, D', B) is true.

Bmin is defined as the smallest value of B' such that Solvable(F, D, B') is true.

**Limits**

1 ≤ N ≤ 100

Small dataset

1 ≤ F ≤ 100

1 ≤ D ≤ 100

1 ≤ B ≤ 100

Large dataset

1 ≤ F ≤ 2000000000

1 ≤ D ≤ 2000000000

1 ≤ B ≤ 2000000000

**Sample**

Input

2

3 3 3

7 5 3

Output

1: 7 2 1

2: 25 3 2

Subject: MATH PROB. - 2 Eggs & A 100 Story Building (S381b) on 5/9/2004

You have a 100 story building and 2 eggs. These are especially strong eggs. There is some floor below which the egg will not break if dropped. What is the worst case upper bound on the number of drops you must make to determine this floor?

THE SOLUTION (Done by my Cousin Ron and me)

Let N the number of drops you need to find the first floor that breaks eggs. Go to the Nth floor and drop an egg. If it breaks you have N - 1 more drops to test the N - 1 floors below. If it doesn't breaks, go to the (N + (N - 1))th floor and drop the same egg. If it breaks you have N -2 drops left to test the N - 2 floors between Nth floor and (2N-1)th floor. By a similar analysis you approach the top of the building with

N + (N - 1) + (N - 2) + (N - 3) + . . . + 1 ≥ 100

N ( N + 1 ) / 2 ≥ 100

This is a quadratic equation which yields N ≥ 13.65. If N = 13 you can only analyze a building of 91 floors. It takes N = 14 to test a 100 floor building.

Final solution. You go to the 14th floor and drop an egg. If it breaks you have 13 more drops to test floors 1 to 13. If it doesn't break you go to the (14 + 13)=27th and drop the first egg again. If it breaks you have 12 drops left to test the 12 floors above 14 and below 27. You continue up the building until you reach the 99th floor (14+13+12+11+10+9+8+7+6+5+4) with 3 drops left. If it breaks you have 3 drops left to test the 3 floors between the 95th floor and the 99th floor.

Subject: MATH PROB. - 3 Eggs & A 1000 Story Building (S387b) on 5/9/2004

During the solution of the math problem "2 Eggs & a 100 Story Building" in Sunday Morning Laughs #381b, I challenged the readers to the next level problem:

You have a 1000 story building and 3 eggs. These are especially strong eggs. There is some floor below which the egg will not break if dropped. What is the worst case upper bound on the number of drops you must make to determine this floor?

THE SOLUTION (Done by Jack)

First let me say I've yet to read the answer for the 2 egg and 100 story building, so I must share how I got to my solution: Before I try to turn something I've not seen before into an equation I like to play with it a bit. In this case I looked at the normal binary approach and saw the general situation that once the egg breaks you must deal with all the unknown floors one at a time, and so the rate that you climb the building must be balanced against the drops required to fill in the gap. Another sampling at 10 floors per drop made it clear that the optimal rate also depended upon the height of the building. Since the height was reduced with each drop, I prepared to solve it recursively when the answer leaped out at me. With a known optimal number for a particular height the next lower height must require one fewer drops, since to get to that lower height you've taken one drop. At the limit of a one story building it takes one drop. As you increase the height, adding one drop each time, it's obvious that this allows a one floor increase in the climb rate. Thus a 100 story building has a minimum of 14 drops - the first number where the summation exceeds 100.

With a 1000 story building and 3 eggs the approach is the same: You want a climb rate that reduces the required number of drops by one each time you take a drop until you reach the lower limit. It seems clear (I wish I could tell you why it's clear - this is why I could never be a teacher) that the step rate will be equal to the summation. The sum of the sums that is equal to or more than 1000 is for n = 18 (it's actually the sum of the sums plus n). So the answer is 19 drops:

N(N^2 + 5)/6 ≥ 1000

N=19

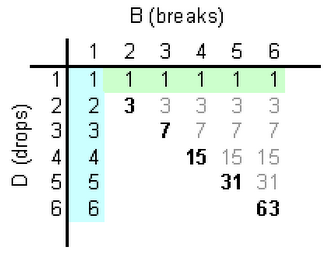
|  |  |  |  |
| --- | --- | --- | --- |
| Drop  Floor | 2 Egg Building Height | Remaining Drops | Total Drops |
| 172 | 171 | 18 | 19 |
| 326 | 153 | 17 | 19 |
| 463 | 136 | 16 | 19 |
| 584 | 120 | 15 | 19 |
| 690 | 105 | 14 | 19 |
| 782 | 91 | 13 | 19 |
| 861 | 78 | 12 | 19 |
| 928 | 66 | 11 | 19 |
| 984 | 55 | 10 | 19 |
| 1030 | 45 | 9 | 19 |
| 1067 | 36 | 8 | 19 |
| 1096 | 28 | 7 | 19 |
| 1118 | 21 | 6 | 19 |
| 1134 | 15 | 5 | 19 |
| 1145 | 10 | 4 | 19 |
| 1152 | 6 | 3 | 19 |
| 1156 | 3 | 2 | 19 |
| 1158 | 1 | 1 | 19 |

Jack, you solution is correct, clear, and the first solution.

Dynamic programming:

1. The main idea is to understand dependency between current drop and next/previous drops.
2. Let *F(D, B)* - function returning *Fmax* number of floors in a building when *Solvable(F, D, B)* is *true*.
3. Let do a drop. We know the result for this drop: *egg has been broken* or *egg has not been broken*. So we covered *+1* floor. Now we have *D-1* drops left.
4. If *egg has not been broken* then we still have *B* breaks left and for all floors from *1 to current* egg will not be dropped as well. To get value of *Fmax* we only need to estimate how many floors upstairs we could cover to have *Solvable(F, D, B)* as *true*. We could think that we are on the ground and required value could be evaluated by *F(D-1, B)*.
5. If *egg has been broken* then we have *B-1* breaks left. The situation below current floor is still unknown but to estimate how many floors downstairs we could cover and have *Solvable(F, D, B)* as *true* we could use *F(D-1, B-1)*.
6. Taking all above into an account we have *F(D, B)* = *1* + *F(D-1, B)* + *F(D-1, B-1)*

Implementation:

1. Next facts could help to have efficient implementation.
2. *F(D, 1) = D* - because to have *Solvable(F, D, B) = true*we should start from 1-st floor and continue dropping from next floor one by one.
3. *F(1, B) = 1* - because to have *Solvable(F, D, B) = true*we should start from 1-st floor and no more drops left after first attempt.
4. *F(x, x) = 2^x -1* - because to have *Solvable(F, D, B) = true* and to have optimal solution we should use *divide and conquer* approach
5. *F(x, y) = F(x, x)*, for all *y > x* - the same as for (4) and more possible breaks don't affect the result.
6. We need to calculate F(D, B) only for B=1..32, because all other values could be evaluated by (5) or will be -1, because they will be greater then 2^32 (4294967296)
7. It is enough to have array[1..Z][1..32], where *F(Z, 2) > 4294967296* values to have fast, cache-based implementation of *F(D, B)*.  
   A general solution for Floor(D, B): <http://mathbin.net/49980>  
   Floor(D, 2) = D\*(D+1)/2  
   Z(Z+1)/2 = 4294967296  
   Z =(approx.) 92681.4

*F(D, B)* - Illustration of facts discussed above