## Q1.1

Braille Translation

===================

Because Commander Lambda is an equal-opportunity despot, she has several visually-impaired minions. But she never bothered to follow intergalactic standards for workplace accommodations, so those minions have a hard time navigating her space station. You figure printing out Braille signs will help them, and - since you'll be promoting efficiency at the same time - increase your chances of a promotion.

Braille is a writing system used to read by touch instead of by sight. Each character is composed of 6 dots in a 2x3 grid, where each dot can either be a bump or be flat (no bump). You plan to translate the signs around the space station to Braille so that the minions under Commander Lambda's command can feel the bumps on the signs and "read" the text with their touch. The special printer which can print the bumps onto the signs expects the dots in the following order:

1 4

2 5

3 6

So given the plain text word "code", you get the Braille dots:

11 10 11 10

00 01 01 01

00 10 00 00

where 1 represents a bump and 0 represents no bump. Put together, "code" becomes the output string "100100101010100110100010".

Write a function answer(plaintext) that takes a string parameter and returns a string of 1's and 0's representing the bumps and absence of bumps in the input string. Your function should be able to encode the 26 lowercase letters, handle capital letters by adding a Braille capitalization mark before that character, and use a blank character (000000) for spaces. All signs on the space station are less than fifty characters long and use only letters and spaces.

Languages

=========

To provide a Python solution, edit solution.py

To provide a Java solution, edit solution.java

Test cases

==========

Inputs:

(string) plaintext = "code"

Output:

(string) "100100101010100110100010"

Inputs:

(string) plaintext = "Braille"

Output:

(string) "000001110000111010100000010100111000111000100010"

Inputs:

(string) plaintext = "The quick brown fox jumped over the lazy dog"

Output:

(string) "000001011110110010100010000000111110101001010100100100101000000000110000111010101010010111101110000000110100101010101101000000010110101001101100111100100010100110000000101010111001100010111010000000011110110010100010000000111000100000101011101111000000100110101010110110"

## Q2.1

Keeping track of Commander Lambda's many bunny prisoners is starting to get tricky. You've been tasked with writing a program to match bunny prisoner IDs to cell locations.

The LAMBCHOP doomsday device takes up much of the interior of Commander Lambda's space station, and as a result the prison blocks have an unusual layout. They are stacked in a triangular shape, and the bunny prisoners are given numerical IDs starting from the corner, as follows:

| 7

| 4 8

| 2 5 9

| 1 3 6 10

Each cell can be represented as points (x, y), with x being the distance from the vertical wall, and y being the height from the ground.

For example, the bunny prisoner at (1, 1) has ID 1, the bunny prisoner at (3, 2) has ID 9, and the bunny prisoner at (2,3) has ID 8. This pattern of numbering continues indefinitely (Commander Lambda has been taking a LOT of prisoners).

Write a function answer(x, y) which returns the prisoner ID of the bunny at location (x, y). Each value of x and y will be at least 1 and no greater than 100,000. Since the prisoner ID can be very large, return your answer as a string representation of the number.

## Q2.2

# Lovely Lucky LAMBs

Being a henchman isn't all drudgery. Occasionally, when Commander Lambda is feeling generous, she'll hand out Lucky LAMBs (Lambda's All-purpose Money Bucks). Henchmen can use Lucky LAMBs to buy things like a second pair of socks, a pillow for their bunks, or even a third daily meal!

However, actually passing out LAMBs isn't easy. Each henchman squad has a strict seniority ranking which must be respected - or else the henchmen will revolt and you'll all get demoted back to minions again!

There are 4 key rules which you must follow in order to avoid a revolt:  
1. The most junior henchman (with the least seniority) gets exactly 1 LAMB. (There will always be at least 1 henchman on a team.)  
2. A henchman will revolt if the person who ranks immediately above them gets more than double the number of LAMBs they do.  
3. A henchman will revolt if the amount of LAMBs given to their next two subordinates combined is more than the number of LAMBs they get. (Note that the two most junior henchmen won't have two subordinates, so this rule doesn't apply to them. The 2nd most junior henchman would require at least as many LAMBs as the most junior henchman.)  
4. You can always find more henchmen to pay - the Commander has plenty of employees. If there are enough LAMBs left over such that another henchman could be added as the most senior while obeying the other rules, you must always add and pay that henchman.

Note that you may not be able to hand out all the LAMBs. A single LAMB cannot be subdivided. That is, all henchmen must get a positive integer number of LAMBs.

Write a function called answer(total\_lambs), where total\_lambs is the integer number of LAMBs in the handout you are trying to divide. It should return an integer which represents the difference between the minimum and maximum number of henchmen who can share the LAMBs (that is, being as generous as possible to those you pay and as stingy as possible, respectively) while still obeying all of the above rules to avoid a revolt. For instance, if you had 10 LAMBs and were as generous as possible, you could only pay 3 henchmen (1, 2, and 4 LAMBs, in order of ascending seniority), whereas if you were as stingy as possible, you could pay 4 henchmen (1, 1, 2, and 3 LAMBs). Therefore, answer(10) should return 4-3 = 1.

To keep things interesting, Commander Lambda varies the sizes of the Lucky LAMB payouts: you can expect total\_lambs to always be between 10 and 1 billion (10 ^ 9).

# Languages

To provide a Python solution, edit solution.py  
To provide a Java solution, edit solution.java

# Test cases

Inputs:  
(int) total\_lambs = 10  
Output:  
(int) 1

Inputs:  
(int) total\_lambs = 143  
Output:  
(int) 3

## Q3.1

# The Grandest Staircase Of Them All

With her LAMBCHOP doomsday device finished, Commander Lambda is preparing for her debut on the galactic stage - but in order to make a grand entrance, she needs a grand staircase! As her personal assistant, you've been tasked with figuring out how to build the best staircase EVER.

Lambda has given you an overview of the types of bricks available, plus a budget. You can buy different amounts of the different types of bricks (for example, 3 little pink bricks, or 5 blue lace bricks). Commander Lambda wants to know how many different types of staircases can be built with each amount of bricks, so she can pick the one with the most options.

Each type of staircase should consist of 2 or more steps. No two steps are allowed to be at the same height - each step must be lower than the previous one. All steps must contain at least one brick. A step's height is classified as the total amount of bricks that make up that step.  
For example, when N = 3, you have only 1 choice of how to build the staircase, with the first step having a height of 2 and the second step having a height of 1: (# indicates a brick)

21

When N = 4, you still only have 1 staircase choice:

31

But when N = 5, there are two ways you can build a staircase from the given bricks. The two staircases can have heights (4, 1) or (3, 2), as shown below:

41

32

Write a function called answer(n) that takes a positive integer n and returns the number of different staircases that can be built from exactly n bricks. n will always be at least 3 (so you can have a staircase at all), but no more than 200, because Commander Lambda's not made of money!

# Languages

To provide a Python solution, edit solution.py  
To provide a Java solution, edit solution.java

# Test cases

Inputs:  
(int) n = 3  
Output:  
(int) 1

Inputs:  
(int) n = 200  
Output:  
(int) 487067745

## Q3.2

# Doomsday Fuel

Making fuel for the LAMBCHOP's reactor core is a tricky process because of the exotic matter involved. It starts as raw ore, then during processing, begins randomly changing between forms, eventually reaching a stable form. There may be multiple stable forms that a sample could ultimately reach, not all of which are useful as fuel.

Commander Lambda has tasked you to help the scientists increase fuel creation efficiency by predicting the end state of a given ore sample. You have carefully studied the different structures that the ore can take and which transitions it undergoes. It appears that, while random, the probability of each structure transforming is fixed. That is, each time the ore is in 1 state, it has the same probabilities of entering the next state (which might be the same state). You have recorded the observed transitions in a matrix. The others in the lab have hypothesized more exotic forms that the ore can become, but you haven't seen all of them.

Write a function answer(m) that takes an array of array of nonnegative ints representing how many times that state has gone to the next state and return an array of ints for each terminal state giving the exact probabilities of each terminal state, represented as the numerator for each state, then the denominator for all of them at the end and in simplest form. The matrix is at most 10 by 10. It is guaranteed that no matter which state the ore is in, there is a path from that state to a terminal state. That is, the processing will always eventually end in a stable state. The ore starts in state 0. The denominator will fit within a signed 32-bit integer during the calculation, as long as the fraction is simplified regularly.

For example, consider the matrix m:  
[  
[0,1,0,0,0,1], # s0, the initial state, goes to s1 and s5 with equal probability  
[4,0,0,3,2,0], # s1 can become s0, s3, or s4, but with different probabilities  
[0,0,0,0,0,0], # s2 is terminal, and unreachable (never observed in practice)  
[0,0,0,0,0,0], # s3 is terminal  
[0,0,0,0,0,0], # s4 is terminal  
[0,0,0,0,0,0], # s5 is terminal  
]  
So, we can consider different paths to terminal states, such as:  
s0 -> s1 -> s3  
s0 -> s1 -> s0 -> s1 -> s0 -> s1 -> s4  
s0 -> s1 -> s0 -> s5  
Tracing the probabilities of each, we find that  
s2 has probability 0  
s3 has probability 3/14  
s4 has probability 1/7  
s5 has probability 9/14  
So, putting that together, and making a common denominator, gives an answer in the form of  
[s2.numerator, s3.numerator, s4.numerator, s5.numerator, denominator] which is  
[0, 3, 2, 9, 14].

# Languages

To provide a Python solution, edit solution.py  
To provide a Java solution, edit solution.java

# Test cases

Inputs:  
(int) m = [[0, 2, 1, 0, 0], [0, 0, 0, 3, 4], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]  
Output:  
(int list) [7, 6, 8, 21]

Inputs:  
(int) m = [[0, 1, 0, 0, 0, 1], [4, 0, 0, 3, 2, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]  
Output:  
(int list) [0, 3, 2, 9, 14]

## Q3.3

# Prepare the Bunnies' Escape

You're awfully close to destroying the LAMBCHOP doomsday device and freeing Commander Lambda's bunny prisoners, but once they're free of the prison blocks, the bunnies are going to need to escape Lambda's space station via the escape pods as quickly as possible. Unfortunately, the halls of the space station are a maze of corridors and dead ends that will be a deathtrap for the escaping bunnies. Fortunately, Commander Lambda has put you in charge of a remodeling project that will give you the opportunity to make things a little easier for the bunnies. Unfortunately (again), you can't just remove all obstacles between the bunnies and the escape pods - at most you can remove one wall per escape pod path, both to maintain structural integrity of the station and to avoid arousing Commander Lambda's suspicions.

You have maps of parts of the space station, each starting at a prison exit and ending at the door to an escape pod. The map is represented as a matrix of 0s and 1s, where 0s are passable space and 1s are impassable walls. The door out of the prison is at the top left (0,0) and the door into an escape pod is at the bottom right (w-1,h-1).

Write a function answer(map) that generates the length of the shortest path from the prison door to the escape pod, where you are allowed to remove one wall as part of your remodeling plans. The path length is the total number of nodes you pass through, counting both the entrance and exit nodes. The starting and ending positions are always passable (0). The map will always be solvable, though you may or may not need to remove a wall. The height and width of the map can be from 2 to 20. Moves can only be made in cardinal directions; no diagonal moves are allowed.

# Languages

To provide a Python solution, edit solution.py  
To provide a Java solution, edit solution.java

# Test cases

Inputs:  
(int) maze = [[0, 1, 1, 0], [0, 0, 0, 1], [1, 1, 0, 0], [1, 1, 1, 0]]  
Output:  
(int) 7

Inputs:  
(int) maze = [[0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 0], [0, 0, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1], [0, 1, 1, 1, 1, 1], [0, 0, 0, 0, 0, 0]]  
Output:  
(int) 11

Use verify [file] to test your solution and see how it does. When you are finished editing your code, use submit [file] to submit your answer. If your solution passes the test cases, it will be removed from your home folder.

## Q4.1

# Bringing a Gun to a Guard Fight

Uh-oh - you've been cornered by one of Commander Lambdas elite guards! Fortunately, you grabbed a beam weapon from an abandoned guardpost while you were running through the station, so you have a chance to fight your way out. But the beam weapon is potentially dangerous to you as well as to the elite guard: its beams reflect off walls, meaning youll have to be very careful where you shoot to avoid bouncing a shot toward yourself!

Luckily, the beams can only travel a certain maximum distance before becoming too weak to cause damage. You also know that if a beam hits a corner, it will bounce back in exactly the same direction. And of course, if the beam hits either you or the guard, it will stop immediately (albeit painfully).

Write a function answer(dimensions, your\_position, guard\_position, distance) that gives an array of 2 integers of the width and height of the room, an array of 2 integers of your x and y coordinates in the room, an array of 2 integers of the guard's x and y coordinates in the room, and returns an integer of the number of distinct directions that you can fire to hit the elite guard, given the maximum distance that the beam can travel.

The room has integer dimensions [1 < x\_dim <= 1000, 1 < y\_dim <= 1000]. You and the elite guard are both positioned on the integer lattice at different distinct positions (x, y) inside the room such that [0 < x < x\_dim, 0 < y < y\_dim]. Finally, the maximum distance that the beam can travel before becoming harmless will be given as an integer 1 < distance <= 10000.

For example, if you and the elite guard were positioned in a room with dimensions [3, 2], you\_position [1, 1], guard\_position [2, 1], and a maximum shot distance of 4, you could shoot in seven different directions to hit the elite guard (given as vector bearings from your location): [1, 0], [1, 2], [1, -2], [3, 2], [3, -2], [-3, 2], and [-3, -2]. As specific examples, the shot at bearing [1, 0] is the straight line horizontal shot of distance 1, the shot at bearing [-3, -2] bounces off the left wall and then the bottom wall before hitting the elite guard with a total shot distance of sqrt(13), and the shot at bearing [1, 2] bounces off just the top wall before hitting the elite guard with a total shot distance of sqrt(5).

# Languages

To provide a Python solution, edit solution.py  
To provide a Java solution, edit solution.java

# Test cases

Inputs:  
(int list) dimensions = [3, 2]  
(int list) captain\_position = [1, 1]  
(int list) badguy\_position = [2, 1]  
(int) distance = 4  
Output:  
(int) 7

Inputs:  
(int list) dimensions = [300, 275]  
(int list) captain\_position = [150, 150]  
(int list) badguy\_position = [185, 100]  
(int) distance = 500  
Output:  
(int) 9