LEVEL 1 PART 1

Re-ID

=====

There's some unrest in the minion ranks: minions with ID numbers like "1",

"42", and other "good" numbers have been lording it over the poor minions who

are stuck with more boring IDs. To quell the unrest, Commander Lambda has tasked you with

reassigning everyone new random IDs based on a Completely Foolproof Scheme.

Commander Lambda has concatenated the prime numbers in a single long string:

"2357111317192329...". Now every minion must draw a number from a hat. That number is the

starting index in that string of primes, and the minion's new ID number will be the next five

digits in the string. So if a minion draws "3", their ID number will be "71113".

Help the Commander assign these IDs by writing a function solution(n) which takes in the starting

index n of Lambda's string of all primes, and returns the next five digits in the string.

Commander Lambda has a lot of minions, so the value of n will always be between 0 and 10000.

Languages

=========

To provide a Java solution, edit Solution.java

To provide a Python solution, edit solution.py

Test cases

==========

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Java cases --

Input:

Solution.solution(0)

Output:

23571

Input:

Solution.solution(3)

Output:

71113

-- Python cases --

Input:

solution.solution(0)

Output:

23571

Input:

solution.solution(3)

Output:

71113

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.

LEVEL 2 PART 1

Power Hungry

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

Commander Lambda's space station is HUGE. And huge space stations take a LOT of power. Huge space

stations with doomsday devices take even more power. To help meet the station's power needs,

Commander Lambda has installed solar panels on the station's outer surface. But the station sits

in the middle of a quasar quantum flux field, which wreaks havoc on the solar panels. You and your

team of henchmen have been assigned to repair the solar panels, but you'd rather not take down

all of the panels at once if you can help it, since they do help power the space station and all!

You need to figure out which sets of panels in any given array you can take offline to repair while

still maintaining the maximum amount of power output per array, and to do THAT, you'll first

need to figure out what the maximum output of each array actually is. Write a function solution(xs)

that takes a list of integers representing the power output levels of each panel in an array, and

returns the maximum product of some non-empty subset of those numbers. So for example, if an array

contained panels with power output levels of [2, -3, 1, 0, -5], then the maximum product would be

found by taking the subset: xs[0] = 2, xs[1] = -3, xs[4] = -5, giving the product 2\*(-3)\*(-5) = 30.

So solution([2,-3,1,0,-5]) will be "30".

Each array of solar panels contains at least 1 and no more than 50 panels, and each panel will have a

power output level whose absolute value is no greater than 1000 (some panels are malfunctioning so

badly that they're draining energy, but you know a trick with the panels' wave stabilizer

that lets you combine two negative-output panels to produce the positive output of the multiple of

their power values). The final products may be very large, so give the solution as a string

representation of the number.

Languages

=========

To provide a Python solution, edit solution.py

To provide a Java solution, edit Solution.java

Test cases

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Python cases --

Input:

solution.solution([2, 0, 2, 2, 0])

Output:

8

Input:

solution.solution([-2, -3, 4, -5])

Output:

60

-- Java cases --

Input:

Solution.solution({2, 0, 2, 2, 0})

Output:

8

Input:

Solution.solution({-2, -3, 4, -5})

Output:

60

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.

LEVEL 2 PART 2

Ion Flux Relabeling

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

Oh no! Commander Lambda's latest experiment to improve the efficiency of the LAMBCHOP doomsday

device has backfired spectacularly. The Commander had been improving the structure of the ion flux

converter tree, but something went terribly wrong and the flux chains exploded. Some of the ion flux

converters survived the explosion intact, but others had their position labels blasted off.

Commander Lambda is having her henchmen rebuild the ion flux converter tree by hand, but you think

you can do it much more quickly -- quickly enough, perhaps, to earn a promotion!

Flux chains require perfect binary trees, so Lambda's design arranged the ion flux converters to

form one. To label them, Lambda performed a post-order traversal of the tree of converters and

labeled each converter with the order of that converter in the traversal, starting at 1. For

example, a tree of 7 converters would look like the following:

7

3 6

1 2 4 5

Write a function solution(h, q) - where h is the height of the perfect tree of converters and q is a

list of positive integers representing different flux converters - which returns a list of integers p

where each element in p is the label of the converter that sits on top of the respective converter

in q, or -1 if there is no such converter. For example, solution(3, [1, 4, 7]) would return the

converters above the converters at indexes 1, 4, and 7 in a perfect binary tree of height 3, which

is [3, 6, -1].

The domain of the integer h is 1 <= h <= 30, where h = 1 represents a perfect binary tree

containing only the root, h = 2 represents a perfect binary tree with the root and two leaf nodes, h

= 3 represents a perfect binary tree with the root, two internal nodes and four leaf nodes (like the

example above), and so forth. The lists q and p contain at least one but no more than 10000

distinct integers, all of which will be between 1 and 2^h-1, inclusive.

Languages

To provide a Java solution, edit Solution.java

To provide a Python solution, edit solution.py

Test cases

==========

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Java cases --

Input:

Solution.solution(3, {7, 3, 5, 1})

Output:

-1,7,6,3

Input:

Solution.solution(5, {19, 14, 28})

Output:

21,15,29

-- Python cases --

Input:

solution.solution(5, [19, 14, 28])

Output:

21,15,29

Input:

solution.solution(3, [7, 3, 5, 1])

Output:

-1,7,6,3

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.

LEVEL 3 PART 1

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

With the LAMBCHOP doomsday device finished, Commander Lambda is preparing to debut on the galactic stage -- but in order to make a grand entrance, Lambda needs a grand staircase! As the Commander's 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 they 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 solution(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 Java solution, edit Solution.java

To provide a Python solution, edit solution.py

Test cases

==========

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Java cases --

Input:

Solution.solution(200)

Output:

487067745

Input:

Solution.solution(3)

Output:

1

-- Python cases --

Input:

solution.solution(200)

Output:

487067745

LEVEL 3 PART 2

Prepare the Bunnies' Escape

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

You're awfully close to destroying the LAMBCHOP doomsday device and freeing Commander

Lambda's bunny workers, but once they're free of the work duties 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 work area 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 station 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 solution(map) that generates the length of the shortest path from the station 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.

Test cases

==========

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Python cases --

Input:

solution.solution([[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:

11

Input:

solution.solution([[0, 1, 1, 0], [0, 0, 0, 1], [1, 1, 0, 0], [1, 1, 1, 0]])

Output:

7

-- Java cases --

Input:

Solution.solution({{0, 1, 1, 0}, {0, 0, 0, 1}, {1, 1, 0, 0}, {1, 1, 1, 0}})

Output:

7

Input:

Solution.solution({{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:

11

LEVEL 3 PART 3

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 solution(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 Java solution, edit Solution.java

To provide a Python solution, edit solution.py

Test cases

==========

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Java cases --

Input:

Solution.solution({{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:

[0, 3, 2, 9, 14]

Input:

Solution.solution({{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:

[7, 6, 8, 21]

-- Python cases --

Input:

solution.solution([[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:

[7, 6, 8, 21]

Input:

solution.solution([[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:

[0, 3, 2, 9, 14]

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.

LEVEL 4 PART 1

Distract the Trainers

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

The time for the mass escape has come, and you need to distract the bunny trainers so that the workers can make it out! Unfortunately for you, they're watching the bunnies closely. Fortunately, this means they haven't realized yet that the space station is about to explode due to the destruction of the LAMBCHOP doomsday device. Also fortunately, all that time you spent working as first a minion and then a henchman means that you know the trainers are fond of bananas. And gambling. And thumb wrestling.

The bunny trainers, being bored, readily accept your suggestion to play the Banana Games.

You will set up simultaneous thumb wrestling matches. In each match, two trainers will pair off to thumb wrestle. The trainer with fewer bananas will betall their bananas, and the other trainer will match the bet. The winner will receive all of the bet bananas. You don't pair off trainers with the samenumber of bananas (you will see why, shortly). You know enough trainer psychology to know that the one who has more bananas always gets over-confident andloses. Once a match begins, the pair of trainers will continue to thumb wrestle and exchange bananas, until both of them have the same number of bananas.

Once that happens, both of them will lose interest and go back to supervising the bunny workers, and you don't want THAT to happen!

For example, if the two trainers that were paired started with 3 and 5 bananas, after the first round of thumb wrestling they will have 6 and 2 (the one with 3 bananas wins and gets 3 bananas from the loser). After the second round, they will have 4 and 4 (the one with 6 bananas loses 2 bananas). At that point they stop and get back to training bunnies.

How is all this useful to distract the bunny trainers? Notice that if the trainers had started with 1 and 4 bananas, then they keep thumb wrestling! 1, 4

-> 2, 3 -> 4, 1 -> 3, 2 -> 1, 4 and so on.

Now your plan is clear. You must pair up the trainers in such a way that the maximum number of trainers go into an infinite thumb wrestling loop!

Write a function solution(banana\_list) which, given a list of positive integers depicting the amount of bananas the each trainer starts with, returns the fewest possible number of bunny trainers that will be left to watch the workers. Element i of the list will be the number of bananas that trainer i (counting from 0) starts with.

The number of trainers will be at least 1 and not more than 100, and the number of bananas each trainer starts with will be a positive integer no more than 1073741823 (i.e. 2^30 -1). Some of them stockpile a LOT of bananas.

Languages

=========

To provide a Python solution, edit solution.py

To provide a Java solution, edit Solution.java

Test cases

==========

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Python cases --

Input:

solution.solution([1, 7, 3, 21, 13, 19])

Output:

0

Input:

solution.solution(1,1)

Output:

2