Distributed Round 2 2017

Submissions

Testrun

0pt	Not attempted **0/58 users** correct (0%)

flagpoles

1pt	Not attempted **335/181 users** correct (185%)
11pt	Not attempted **277/320 users** correct (87%)

number_bases

5pt	Not attempted **241/186 users** correct (130%)
17pt	Not attempted **188/226 users** correct (83%)

broken_memory

3pt	Not attempted **196/88 users** correct (223%)
25pt	Not attempted **77/142 users** correct (54%)

nanobots

8pt	Not attempted **104/69 users** correct (151%)
30pt	Not attempted **31/68 users** correct (46%)

Top Scores

fagu	100
bmerry	100
krijgertje	100
ecnerwala	100
pashka	100
Swistakk	100
KalininN	100
adsz	100
Gennady.Korotkevich	100
eatmore	100

Problem A. Testrun

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small
0 points
2 minute timeout

The contest is finished.

Problem

This is a way to test your solutions, not a real problem!

When you submit a solution to this problem, it will run one testcase on a 100 nodes. This will allow you to estimate how fast your solution will run on our system.

Remember to change your solution appropriately before submitting it for real, so you don't fail because of a compilation error! The best way to check is to run your solution on the small input before submitting to the large input.

Input

There is no input for this problem. This means you should not include / import an input library.

Output

Doesn't really matter what you output. If your solution runs successfully to completion, it will be judged as "Wrong Answer".

Limits

Each node will have access to 1 GB of RAM, and a time limit of 26 seconds. The maximum number of messages a single node can send is 5000, and the maximum sum of the sizes of those messages is 8MB.
This problem only has one small test case. It will run on 100 nodes.

Problem B. flagpoles

small 1 points *2 minute timeout*	The contest is finished.
large 11 points *10 minute timeout*	The contest is finished.

Problem

Flagpoles

Cody-Jamal, the famous conceptual artist, was called to design the new United Nations headquarters. The entrance displays a single row of flagpoles with the flags of different countries. Each flagpole is exactly 1 meter away from its neighbor(s). Since different nations have different rules about how high their flags must be flown, the tips of the flagpoles may have different heights.

The scientists from the famous Detecting Collinearity Journal have become interested in the flagpoles. In particular, they want to know the maximum number of consecutive flagpoles with tips that are collinear. A set of contiguous flagpoles has collinear tips if there is a constant d such that, for every pair of adjacent flagpoles in the set, the height of the right flagpole's tip minus the height of the left flagpole's tip is equal to d. Notice that the condition is always true for a set of up to 2 flagpoles.

For example, if the flagpoles' heights are 5, 7, 5, 3, 1, 2, 3, in left-to-right order, the leftmost 2 flagpoles and the rightmost 3 flagpoles are examples of consecutive sets of flagpoles with collinear tips. The flagpoles with heights 7 and 1, together with those in between them, are another example. The leftmost 3 flagpoles, however, do not have collinear tips, so they do not form such a set.

Given the height in meters of each flagpole tip, in the left-to-right order in which they appear, can you help the DCJ calculate the maximum size of a set of consecutive flagpoles with collinear tips?

Input

The input library is called "flagpoles"; see the sample inputs below for examples in your language. It defines two methods:

- **GetNumFlagpoles()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of flagpoles in the row.
  - Expect each call to take 0.17 microseconds.
- **GetHeight(i)**:
  - Takes exactly one 64-bit integer argument: a position i, 0 ≤ i < `GetNumFlagpoles()`.
  - Returns a 64-bit integer: the height, in meters, of the flagpole at the ith position from left to right. The ith flagpole is always i meters to the right of the 0th flagpole.
  - Expect each call to take 0.17 microseconds.

Output

Output one line with a single integer: the maximum number of consecutive flagpoles with collinear top ends.

Limits

Time limit: 3 seconds.
Memory limit per node: 512 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$1 \le \text{GetHeight(i)} \le 10^{18}$.

Small dataset

Number of nodes: 10.
$1 \le \text{GetNumFlagpoles()} \le 10^{6}$.

Large dataset

Number of nodes: 100.
$1 \le \text{GetNumFlagpoles()} \le 10^{9}$.

Sample

Input	Output
See input files below.	For sample input 1: 4 For sample input 2: 4 For sample input 3: 2

Sample input 1 is the example given in the problem statement.

Sample input libraries:
Sample input for test 1: flagpoles.h [CPP] flagpoles.java [Java]
Sample input for test 2: flagpoles.h [CPP] flagpoles.java [Java]
Sample input for test 3: flagpoles.h [CPP] flagpoles.java [Java]

Problem C. number_bases

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 5 points *2 minute timeout*	The contest is finished.
large 17 points *10 minute timeout*	The contest is finished.

Problem

Number Bases

You are given three sequences X, Y, and Z of equal length. The sequences consist of digits. In this problem we deal with non-decimal bases, so digits are arbitrary non-negative integers, not necessarily restricted to the usual 0 through 9 range.

Your task is to investigate the possible bases for which the equation **X + Y = Z** is both valid and holds true. You must determine whether there is no such base, more than one such base, or only one such base. If there is only one such base, you must find it.

A base is an integer greater than or equal to 2. For X + Y = Z to be valid in base B, all digits in all three sequences have to be strictly less than B. For X + Y = Z to be true in base B, the sum of the integer denoted by X in base B and the integer denoted by Y in base B has to be equal to the integer denoted by Z in base B.

More formally: let S[i] be the i-th digit from the right of a sequence of digits S, with i counted starting from 0. Then, for a given S and an integer base B, we will define the integer denoted by S in base B as f(S, B) = the sum of all S[i] × B^i. Then the equation X + Y = Z is true for a base B if and only if f(X, B) + f(Y, B) = f(Z, B).

For example, consider the sequences X = {1, 2, 3}, Y = {4, 5, 6} and Z = {5, 8, 0}, written with the most significant digits on the left, as usual. That is, X[0] = 3, X[1] = 2 and X[2] = 1. B = 8 is an invalid base, because it is not strictly greater than the middle digit of Z. B = 10 is a valid base, but the expression is not true in that case because 123 + 456 ≠ 580. B = 9 is a valid base that also makes the expression true, because {1, 2, 3} in base 9 is 102, {4, 5, 6} in base 9 is 375 and {5, 8, 0} in base 9 is 477, and 102 + 375 = 477. For this case, B = 9 is the only possible choice of a valid base B that makes the expression true.

On the other hand, the one-digit sequences X = {10}, Y = {20} and Z = {30} have multiple bases for which the equation X + Y = Z is both valid and true. Any value of B greater than 30 would suffice.

Input

The input library is called "number_bases"; see the sample inputs below for examples in your language. It defines four methods:

- **GetLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of digits in each digit sequence X, Y, and Z.
  - Expect each call to take 0.34 microseconds.
- **GetDigitX(i)**:
  - Takes a 64-bit integer in the range 0 ≤ i < GetLength().
  - Returns a 64-bit integer: the i-th digit in the digit sequence X, numbered from right (least significant) to left (most significant). That is, GetDigitX(0) is the least significant digit of X and GetDigitX(GetLength() - 1) is the most significant digit of X.
  - Expect each call to take 0.34 microseconds.
- **GetDigitY(i)**:
  - Takes a 64-bit integer in the range 0 ≤ i < GetLength().
  - Returns a 64-bit integer: the i-th digit in the digit sequence Y, numbered from right (least significant) to left (most significant).
  - Expect each call to take 0.34 microseconds.
- **GetDigitZ(i)**:
  - Takes a 64-bit integer in the range 0 ≤ i < GetLength().
  - Returns a 64-bit integer: the i-th digit in the digit sequence Z, numbered from right (least significant) to left (most significant).
  - Expect each call to take 0.34 microseconds.

Output

Output a single line with a single token x. If there is a single base B that makes the expression valid and true, x must be the base 10 representation of B. If there are multiple values of B that make the expression valid and true, x must be `NON-UNIQUE`. If there is no such value of B, x must be `IMPOSSIBLE`.

Limits

Time limit: 3 seconds.
Memory limit per node: 128 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$0 \le$ GetDigitX(i) $\le 10^6$, for all i.
$0 \le$ GetDigitY(i) $\le 10^6$, for all i.
$0 \le$ GetDigitZ(i) $\le 10^6$, for all i.
(Contrary to typical notation, it is possible for any of the digit sequences in the input to have a zero at its most significant position. Valid bases are not restricted to be less than 10^6 or any other upper bound.)

Small dataset

Number of nodes: 10.
$1 \le$ GetLength() $\le 10^6$.

Large dataset

Number of nodes: 100.
$1 \le$ GetLength() $\le 10^8$.

Sample

Input	Output
See input files below.	For sample input 1: 9 For sample input 2: NON-UNIQUE For sample input 3: IMPOSSIBLE

Sample input libraries:
Sample input for test 1: number_bases.h [CPP] number_bases.java [Java]
Sample input for test 2: number_bases.h [CPP] number_bases.java [Java]
Sample input for test 3: number_bases.h [CPP] number_bases.java [Java]

Problem D. broken_memory

small 3 points *2 minute timeout*	The contest is finished.
large 25 points *10 minute timeout*	The contest is finished.

Problem

Broken Memory

As you may remember from last year, we have a tendency to screw things up at the worst possible moment. For 2017, we promised ourselves that we wouldn't misplace any test cases, though, and we delivered. Unfortunately, last night we spilled a peach smoothie over some of our Google Cloud servers. We immediately called the Dedicated Cloud Janitor (DCJ), but he was fed up with our continuous messes and refused to help. So, we are turning to our most powerful allies, our contestants, to once again rescue us from ourselves.

Fortunately, the servers were already loaded with the data for the problems, so before the damage, the memory was exactly the same in all of them. We conducted a preliminary investigation that revealed that every node's memory was damaged in a *different* place.

We have narrowed the search to a relatively small part of the memory, and we have encoded that as a list of integers for your convenience. Each node has the same list of integers, except at exactly one damaged position; the value there will be *different* from the corresponding value on all other nodes.

For example, suppose the original data is represented by the following list of integers: 1 5 9 3 1. Suppose the broken index for node 0 is 2. That means that when requesting the values for indices 0, 1, 3 and 4, the returned value will be correct (1, 5, 3 and 1, respectively). However, when requesting the value for index 2 on node 0, the returned value could be 1 or 5 or 352462352, for example, but definitely not the correct one (9). If, on the other hand, node 1's broken index is 0, then, when requesting the value of indices 1, 2, 3 and 4, the correct values (5, 9, 3 and 1, respectively) will be returned, but when requesting index 0, the returned value could be 9 or 5 or 379009, but definitely not the correct one (1). Notice that for node 3, neither index 2 nor index 0 can be the broken index, because each node is broken at a different index from all other nodes.

Can you find the broken index on each node for us?

Input

The input library is called "broken_memory"; see the sample inputs below for examples in your language. It defines two methods:

- **GetLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of values in the part of the memory where all the damage happened.
  - Expect each call to take 0.02 microseconds.
- **GetValue(i)**:
  - Takes a 64-bit number in the range 0 ≤ i < GetLength()
  - Returns a 64-bit integer: the i-th value in the memory.
  - Expect each call to take 0.02 microseconds.

Output

Output a single line with `NumberOfNodes()` integers: the broken index for each node, in ascending order of node ID.

Limits

Time limit: 2 seconds.
Memory limit per node: 256 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 128 KB. **(Notice that this is less than usual.)**
$1 \le \text{GetValue(i)} \le 10^{18}$, for all i.

Small dataset

Number of nodes: 10.

10 ≤ GetLength() ≤ 1000.

Large dataset

Number of nodes: 100.
100 ≤ GetLength() ≤ 10^7.

In this problem, the sample inputs are valid for running on 10 nodes, and the sample outputs are computed using 10 nodes.

Sample

Input	Output
See input files below.	For sample input 1: 0 1 2 3 4 5 6 7 8 9 For sample input 2: 29 28 27 26 25 24 23 22 21 20 For sample input 3: 12 13 14 15 8 9 10 11 4 5

Sample input libraries:
Sample input for test 1: broken_memory.h [CPP] broken_memory.java [Java]
Sample input for test 2: broken_memory.h [CPP] broken_memory.java [Java]
Sample input for test 3: broken_memory.h [CPP] broken_memory.java [Java]

Problem E. nanobots

small 8 points *2 minute timeout*	The contest is finished.
large 30 points *10 minute timeout*	The contest is finished.

Problem

Nanobots

A group of medical researchers is working on a new treatment to fight bacteria. In the treatment, special nanobots are transferred into the patient's body, where they locate and trap any harmful bacteria. This allows the patient to get better, and the researchers can later retrieve the trapped bacteria for further study.

Not any nanobot can trap any bacterium, though. A nanobot can be characterized by two traits: size and speed. A nanobot can only trap bacteria that are smaller than it (otherwise, the bacteria will not fit in the nanobot's cage) and also slower than it (otherwise, the bacteria can escape the nanobot). Formally, a nanobot with speed A and size B can trap a bacterium with speed C and size D if and only if A > C and B > D. The speeds and sizes of both nanobots and bacteria are in the inclusive range [1, `GetRange()`].

You have a group of nanobots and you want to know how effective they are at trapping bacteria. Your goal is to find how many of the `GetRange()`^{2} possible bacteria get trapped by the team of nanobots. Unfortunately, you cannot directly examine the speed and size of your nanobots. You can only experiment by introducing a bacterium with a specific size and speed, and watching whether this bacterium gets trapped by the team of nanobots or not. A team of nanobots will trap a bacterium with speed C and size D if and only if there is at least one nanobot in the team that has both speed strictly greater than C and size strictly greater than D.

You may carry out as many experiments of this sort as you want... within the allowed running time for the problem, of course! You can choose the speed and size of the bacteria in each experiment, and for each one, you receive one piece of data: whether or not the bacteria was trapped. Each experiment uses the full team of nanobots, and the same nanobot can catch bacteria in different experiments. Based on that information, you need to determine how many of the `GetRange()`^{2} possible bacteria would be trapped by the team of nanobots. (Because the speed can take any integer value in [1, `GetRange()`], and the same is true for the size, there are `GetRange()`^{2} possible bacteria.) Since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime 10^9+7 (1000000007).

Distributed Code Jam is not a licensed physician. Nothing in this problem statement should be construed as an attempt to offer medical advice. Distributed Code Jam is also not a licensed scientist. Nothing in this problem statement should be construed as an attempt to offer scientific advice. Using nanobots to fight harmful bacteria definitely sounds cool, though.

Input

The input library is called "nanobots"; see the sample inputs below for examples in your language. It defines three methods:

- **GetNumNanobots()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of nanobots on the team.
  - Expect each call to take 0.2 microseconds.
- **GetRange()**:
  - Takes no argument.
  - Returns a 64-bit integer: the maximum valid value for speeds and sizes of both bacteria and nanobots.
  - Expect each call to take 0.2 microseconds.
- **Experiment(c, d)**:
  - Takes exactly two 64-bit integer arguments: a size c and a speed d, 1 ≤ c,d ≤ `GetRange()`.
  - Returns a char: T if a bacteria with size c and speed d is trapped by the nanobots, or E if it escapes.
  - Expect each call to take 0.2 microseconds.

Output

Output one line with a single integer: how many of the different bacteria with both size and speed in the inclusive range [1, `GetRange()`] would be trapped by the nanobots, modulo the prime 10^9+7 (1000000007). Two bacteria are considered different if and only if they have different speed and/or different size.

Limits

Time limit: 18 seconds.
Memory limit per node: 256 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
$2 \le$ GetRange() $\le 10^{12}$.
`Experiment(c, d)` is either uppercase T or uppercase E, for all c, d.
The results of `Experiment` are consistent with the same team of `GetNumNanobots()` nanobots across all nodes.
It is possible that multiple nanobots might have the same size and speed.

Small dataset

Number of nodes: 10.
$1 \le$ `GetNumNanobots()` $\le 10^5$.

Large dataset

Number of nodes: 100.
$1 \le$ `GetNumNanobots()` $\le 10^7$.

Sample

Input	Output
See input files below.	For sample input 1: 26 For sample input 2: 998496508 For sample input 3: 0

Sample input libraries:
Sample input for test 1: nanobots.h [CPP] nanobots.java [Java]
Sample input for test 2: nanobots.h [CPP] nanobots.java [Java]
Sample input for test 3: nanobots.h [CPP] nanobots.java [Java]