Distributed Round 1 2017

Submissions

Testrun
0pt | Not attempted **0/327 users** correct (0%)

pancakes
2pt | Not attempted **984/406 users** correct (242%)
11pt | Not attempted **920/975 users** correct (94%)

weird_editor
3pt | Not attempted **859/434 users** correct (198%)
20pt | Not attempted **505/807 users** correct (63%)

todd_and_steven
1pt | Not attempted **718/365 users** correct (197%)
30pt | Not attempted **230/437 users** correct (53%)

query_of_death
4pt | Not attempted **483/262 users** correct (184%)
29pt | Not attempted **230/377 users** correct (61%)

Top Scores

mk.al13n	100
semiexp.	100
qwerty787788	100
EgorKulikov	100
ikatanic	100
ecnerwala	100
Golovanov399	100
fagu	100
eatmore	100
Errichto.rekt	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. pancakes

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

Problem

Pancakes

At the Infinite House of Pancakes, **D** hungry diners are seated around a circular table. They are numbered from 0 at the top of the table and clockwise around to **D**-1. Each diner likes a particular type of pancake: the type with the same number as the diner.

You are a pancake server with a stack of pancakes. You are currently at the top of the table (just about to serve diner 0), and you will walk around the table clockwise, perhaps multiple times. Every time you pass by a diner, if the pancake at the top of the stack is the diner's preferred kind, you will serve that pancake. You will keep serving that diner until the top of the stack no longer matches the diner's preference; then you will move on to the next diner, and so on.

Each time you complete a revolution, after passing by diner **D**-1 and before reaching diner 0 again, you check to see if the stack is empty. If so, you are done. Otherwise, you continue clockwise around the table for another revolution, serving the diners.

For example, suppose there are 4 diners and 4 pancakes in the stack 3, 1, 2, 0 in order from top to bottom. You start your first revolution by skipping diners 0, 1 and 2 to serve diner 3 the top pancake, getting the stack down to 1, 2, 0. After that, you go back to diner 0 (starting a second revolution), skipping it to get to diner 1 and giving her pancake 1, getting the stack down to 2, 0. Since she doesn't like pancake 2, you go to diner 2, which gets it and you have only a single pancake 0 left. To serve it, you go through diner 3 and back to 0 (starting a third revolution), who gets the final pancake, and your job is done. You required 3 revolutions in total for this particular arrangement, even though you didn't need to finish the last one completely.

Note that there may be some diners who do not receive any pancakes.

Given the initial stack of pancakes, how many revolutions will you make around the table?

Input

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

- **GetStackSize()**:
  - Takes no argument.
  - Returns a 64-bit integer: the initial number of pancakes in the stack.
  - Expect each call to take 0.8 microseconds.
- **GetNumDiners()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number **D** of diners seated around the table.
  - Expect each call to take 0.8 microseconds.
- **GetStackItem(i)**:
  - Takes a 64-bit number in the range $0 \le i <$ GetStackSize().
  - Returns a 64-bit integer: the i-th pancake type in the stack, where i = 0 corresponds to the first one we plan to serve.
  - Expect each call to take 0.8 microseconds.

Output

Output a single integer: the number of revolutions we make around the table.

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.
$0 \le$ GetStackItem(i) $<$ GetNumDiners().

Small dataset

Number of nodes: 10.
$1 \le$ GetStackSize() $\le 10^5$.
$3 \le$ GetNumDiners() $\le 10^6$.

Large dataset

Number of nodes: 100.
$1 \le$ GetStackSize() $\le 10^8$.
$3 \le$ GetNumDiners() $\le 10^9$.

Sample

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

The first sample case is the one explained in the statement.
Sample input libraries:
Sample input for test 1: pancakes.h [CPP] pancakes.java [Java]
Sample input for test 2: pancakes.h [CPP] pancakes.java [Java]
Sample input for test 3: pancakes.h [CPP] pancakes.java [Java]

Problem C. weird_editor

small 3 points *2 minute timeout*	The contest is finished.
large 20 points *10 minute timeout*	The contest is finished.

Problem

Weird Editor

You just installed a text editor in your computer to edit a text file containing only a positive integer (in base 10). Unfortunately, the editor you installed was not as versatile as you would have hoped.

The editor supports only one operation: choosing and removing any digit, and concatenating one `0` (zero) at the right end of the sequence. In this way, the length of the digit sequence is always preserved.

For instance, suppose your initial digit sequence is `3001`. If you applied the operation to the third digit from the left, you would obtain `3010`. If you then applied the operation on `3010` to the second digit form the left, you would obtain `3100`. In this case, `3100` is the largest result that can be obtained using the allowed operation zero or more times.

What is the maximum number that can be the result of applying the allowed operation to the given input number zero or more times? 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).

Input

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

- **GetNumberLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of digits in the number you're given.
  - Expect each call to take 0.11 microseconds.
- **GetDigit(i)**:
  - Takes a 64-bit integer in the range $0 \le i <$ GetNumberLength().
  - Returns a 64-bit integer: The i-th digit in the given number. Digits are numbered from left (most significant) to right (least significant). That is, GetDigit(0) is the most significant digit and GetDigit(GetNumberLength() - 1) is the least significant digit.
  - Expect each call to take 0.11 microseconds.

Output

Output a single integer: the maximum number that can be obtained by applying the allowed operation to the input number zero or more times. Output that number modulo the prime 10^9+7 (1000000007).

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.
$0 \le$ GetDigit(i) ≤ 9, for all i.
GetDigit(0) $\ne 0$.

Small dataset

Number of nodes: 10.
$2 \le$ GetNumberLength() $\le 10^6$.

Large dataset

Number of nodes: 100.
$2 \le$ GetNumberLength() $\le 10^9$.

Sample

Input	Output
See input files below.	For sample input 1: 3100 For sample input 2: 33000000 For sample input 3: 999999944

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

Problem D. todd_and_steven

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

Problem

Todd and Steven

By now, it is a programming interview cliché: How do you sort a very large sequence of unique integers in increasing order? Finally, we are ready to reveal the correct answer:

1. Give all of the odd integers from the sequence to one assistant named Todd, and ask Todd to sort those integers in increasing order.
2. Give all of the even integers from the sequence to another assistant named Steven, and ask Steven to sort those integers in increasing order.
3. Merge Todd's sequence with Steven's sequence to produce the final sorted sequence.

Todd and Steven have already performed steps 1 and 2 on a certain sequence, and now we would like you to perform step 3. Since the resulting sorted sequence can be very long, we only ask you to output a hash of it as proof that you found it. Let X_j denote the j-th element (counting starting from 0) of the merged sequence. Please find the sum, over all j, of (X_j) XOR j. (Here XOR refers to the bitwise operation between the two integers, represented in both C++ and Java by the operator ^.) 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).

Input

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

- **GetToddLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of values in Todd's sorted sequence.
  - Expect each call to take 0.05 microseconds.
- **GetToddValue(i)**:
  - Takes a 64-bit integer argument in the range 0 ≤ i < `GetToddLength()`.
  - Returns a 64-bit integer: the i-th value of Todd's sorted sequence.
  - Expect each call to take 0.05 microseconds.
- **GetStevenLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the number of values in Steven's sorted sequence.
  - Expect each call to take 0.05 microseconds.
- **GetStevenValue(i)**:
  - Takes a 64-bit integer argument in the range 0 ≤ i < `GetStevenLength()`.
  - Returns a 64-bit integer: the i-th value of Steven's sorted sequence.
  - Expect each call to take 0.05 microseconds.

Output

Output one line with a single 64-bit integer: the sum described in the problem statement, modulo the prime 10^9+7 (1000000007).

Limits

Time limit: 4 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.
$1 \le$ `GetToddValue(i)` $\le 5 \times 10^9$, for all i.
`GetToddValue(i)` < `GetToddValue(i + 1)`, for all i. (Todd's sequence is sorted in increasing order.)
`GetToddValue(i) % 2 = 1`, for all i. (All elements in Todd's sequence are odd.)

$1 \le$ `GetStevenValue(i)` $\le 5 \times 10^9$, for all i.
`GetStevenValue(i)` < `GetStevenValue(i + 1)` for all i. (Steven's sequence is sorted in increasing order.)
`GetStevenValue(i) % 2 = 0`, for all i. (All elements in Steven's sequence are even.)

Small dataset

Number of nodes: 10.
$1 \le$ `GetToddLength()` $\le 10^6$.
$1 \le$ `GetStevenLength()` $\le 10^6$.

Large dataset

Number of nodes: 100.
$1 \le$ `GetToddLength()` $\le 10^9$.
$1 \le$ `GetStevenLength()` $\le 10^9$.

Sample

Input	Output
See input files below.	For sample input 1: 8 For sample input 2: 200 For sample input 3: 111

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

Problem E. query_of_death

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 4 points *2 minute timeout*	The contest is finished.
large 29 points *10 minute timeout*	The contest is finished.

Problem

Query of Death

We planned a nice simple warm-up DCJ problem for you this year: find the sum of many values. You can call a `GetLength()` function to get the number of values and a `GetValue(i)` function to get the i-th of those values; to make it even easier, each of those values is either 0 or 1. Simple, right? Unfortunately, we have been having a technical difficulty, and now the contest is starting and it is too late to fix it.

The issue is that there is exactly one value of i — we are not sure what that value is, but we will call it i_{qod} — that is a "query of death" (a term occasionally used at Google for a query with severe adverse effects) that causes the following malfunction. The first time that GetValue(i_{qod}) is called on a node, the function will return the correct i_{qod}-th value. However, this will cause the GetValue function to "break" on that node. After that, *every* future call to GetValue(i) on that node will return 0 or 1 purely at (pseudo)random, independently of the value of i or of any previous calls. Other nodes are not affected when a node breaks in this way, but the malfunction can still happen in the future: any other node on which you call GetValue(i_{qod}) will also break.

The i_{qod} value that causes the breakage is the same for every node within a test case; it may vary across test cases, though. Nodes do not remain broken across different test cases.

As an example, suppose that we have two unbroken nodes A and B, and two values i_{ok} and i_{qod}. Then the following sequence of calls would produce the following results:

1. GetValue(i_{ok}) on node A: the correct value is returned.
2. GetValue(i_{qod}) on node A: the correct value is returned, but node A breaks.
3. GetValue(i_{ok}) on node B: the correct value is returned.
4. GetValue(i_{ok}) on node A: a random value is returned.
5. GetValue(i_{qod}) on node A: a random value is returned.
6. GetValue(i_{qod}) on node B: the correct value is returned, but node B breaks.
7. GetValue(i_{qod}) on node B: a random value is returned.
8. GetValue(i_{ok}) on node B: a random value is returned.
9. GetValue(i_{qod}) on node A: a random value is returned.
10. GetValue(i_{ok}) on node A: a random value is returned.

We apologize for the inconvenience, but can you find the sum anyway?

Input

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

- **GetLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the total number of values to be summed up. (This function still works correctly even on a broken node.)
  - Expect each call to take 0.2 microseconds.
- **GetValue(i)**:
  - Takes a 64-bit number in the range $0 \le i <$ `GetLength()`.
  - Returns a 32-bit number (which is always either 0 or 1): the `i`-th value if the node is not broken, or 0 or 1 at (pseudo)random if the node is broken.
  - Expect each call to take 0.2 microseconds.

Output

Output a single line with one integer: the sum of all of the values.

Limits

Time limit: 2 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.
There is exactly one i_{qod} value, which is the same for each node, and it is within the allowed range for GetLength().
$0 \le$ GetValue(i) ≤ 1, for all i.

Small dataset

Number of nodes: 10.
$1 \le$ GetLength() $\le 10^4$.

Large dataset

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

Sample

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

The code for the samples simulates the node-breaking behavior described in the statement; the actual test cases have the specified behavior, but the implementation (e.g., of randomness on a broken node) is not necessarily the same.

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