Distributed Practice Round 2015

Submissions

Testrun
0pt | Not attempted **0/142 users** correct (0%)

sandwich
1pt | Not attempted **187/205 users** correct (91%)
15pt | Not attempted **141/178 users** correct (79%)

majority
1pt | Not attempted **170/176 users** correct (97%)
20pt | Not attempted **80/167 users** correct (48%)

shhhh
1pt | Not attempted **110/115 users** correct (96%)
30pt | Not attempted **69/102 users** correct (68%)

load_balance
2pt | Not attempted **94/101 users** correct (93%)
35pt | Not attempted **33/88 users** correct (38%)

Top Scores

iwi	105
simonlindholm	105
Murphy	105
stgatilov	105
Alexander86	105
microtony	105
eatmore	105
uwi	105
Marcin.Smulewicz	105
tczajka	105

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. sandwich

small 1 points *2 minute timeout*	The contest is finished.
large 15 points *10 minute timeout*	The contest is finished.

Problem

Your friends made a brave attempt to beat the world record for making the longest sub sandwich ever. They failed or succeeded (it doesn't really matter), and proposed to you to eat as much of the sandwich as you want to.
The sandwich is composed of N parts. Each part has a taste value for you, which might be positive (meaning you want to eat it) or negative (meaning you would prefer not to). Ideally, you would just eat the tasty parts, but it's rude to break out the middle of the sandwich. So, instead you want to eat some part from the beginning, and some part from the end; and to make the total taste of what you eat as large as possible. The total taste is the sum of tastes of all the parts you have eaten.
Note: it's OK to eat the whole sandwich, or to eat nothing at all. Output the largest total taste of what you can eat.

Input

The input library will be called "sandwich", see the sample inputs below for examples in your language. It will define two methods: GetN(), which will return the number of parts of the sandwich, and GetTaste(i), which will return the taste of the *i*th part of the sandwich, for $0 \le i < N$.
A single call to GetTaste(i) will take approximately 0.01 microseconds.

Output

Output one number: the maximum possible total taste of the parts you will eat.

Limits

Each node will have access to 128MB of RAM, and a time limit of 3 seconds.
$-10^9 \le$ GetTaste(i) $\le 10^9$ for all i with $0 \le i <$ GetN().

Small input

Your solution will run on 10 nodes.
$1 \le$ GetN() ≤ 1000.

Large input

Your solution will run on 100 nodes.
$1 \le$ GetN() $\le 5 \times 10^8$.

Sample

Input	Output
See the input files below.	For sample input 1: 14 For sample input 2: 0 For sample input 3: 5

Note: this problem idea was used by us in the practice round of the Algorithmic Engagements contest in 2014, and also displayed as an example in our FAQ.

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

Distributed Practice Round 2015

A. Testrun
B. sandwich
C. majority
D. shhhh
E. load_balance

Contest Analysis
Questions asked 17

Submissions

Testrun
0pt | Not attempted **0/142 users** correct (0%)

sandwich
1pt | Not attempted **187/205 users** correct (91%)
15pt | Not attempted **141/178 users** correct (79%)

majority
1pt | Not attempted **170/176 users** correct (97%)
20pt | Not attempted **80/167 users** correct (48%)

shhhh
1pt | Not attempted **110/115 users** correct (96%)
30pt | Not attempted **69/102 users** correct (68%)

load_balance
2pt | Not attempted **94/101 users** correct (93%)
35pt | Not attempted **33/88 users** correct (38%)

Top Scores

iwi	105
simonlindholm	105
Murphy	105
stgatilov	105
Alexander86	105
microtony	105
eatmore	105
uwi	105
Marcin.Smulewicz	105
tczajka	105

Problem C. majority

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 20 points *10 minute timeout*	The contest is finished.

Problem

Your country is electing its president, and you are in charge of the new electronic voting system. The citizens have voted, and now you have to check if any of the candidates obtained a *majority* — that is, if there is a candidate for whom more than half of the citizens voted.

Input

The input library will be called "majority", see the sample inputs below for examples in your language. It will define two methods: GetN(), which will return the number of voting citizens *N*, and GetVote(i), which will (for $0 \le i < N$) return the identifier of the candidate for whom citizen *i* voted.

Output

If any candidate obtained a majority of the votes, output the identifier of that candidate. Otherwise, output the string "NO WINNER" (quotes for clarity only). A single call to GetVote(i) will take approximately 0.025 microseconds.

Limits

Each node will have access to 128MB of RAM, and a time limit of 3 seconds.
$0 \le$ GetVote(i) $\le 10^9$ for all *i* with $0 \le i < N$.

Small input

Your solution will run on 10 nodes.
$1 \le$ GetN() ≤ 1000.

Large input

Your solution will run on 100 nodes.
$1 \le$ GetN() $\le 10^9$.

Sample

Input	Output
See the input files below.	For sample input 1: 7 For sample input 2: NO WINNER For sample input 3: NO WINNER

Note: the same problem idea was used by us in a tutorial in the Algorithmic Engagements contest in 2014.

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


Powered by
Google Cloud Platform

Problem D. shhhh

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

Problem

You are at a reception. Along with a (huge) number of other people, you are seated at a round table, and listen to the speaker speak... and speak... and speak. You would like the speech to end, but that's not likely to happen, so you'd at least like to tell your friend — who sits somewhere else at the same table — how badly bored you are.
However, it's rude to speak loudly. So, you'll whisper to one of your two neighbours, asking them to pass the message along. They'll then whisper to the other neighbour, and so on, until the message reaches your friend. You now need to decide which neighbour to choose so the distance traveled by the message is as short as possible, and how long is it going to go.

Input

Each person at the table has a unique integer assigned, from 0 to N-1, where N is the number of people at the table (including you and your friend). You are assigned the identifier 0, and your friend is assigned the identifier 1.
The input library will be called "shhhh", see the sample inputs below for examples in your language. It will define three methods: GetN(), which will return the number N of people at the table, GetLeftNeighbour(i) for $0 \le i < N$, which will return the identifier of the left neighbour of the person with identifier i, and GetRightNeighbour(i) for $0 \le i < N$, which will return the identifier of the right neighbour of the person with identifier i.
A single call to GetLeftNeighbour(i) or GetRightNeighbour(i) will take approximately 0.015 microseconds.

Output

Output one line, containing one word and one number, separated by a single space. The word should be "LEFT" if it is faster to pass the message to your left neighbour, "RIGHT" if it is faster to pass the message to your right neighbour, or "WHATEVER" if the distance is the same in both directions (quotes around all words are for clarity only). The number should be the distance the message will have to travel (that is, the number of people who will hear the message, including your friend, but not including you).

Limits

Each node will have access to 128MB of RAM, and a time limit of 4 seconds.

Small input

Your solution will run on 10 nodes.
$2 \le \text{GetN}() \le 10^7$.

Large input

Your solution will run on 100 nodes.
$2 \le \text{GetN}() \le 10^9$.

Sample

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

Note: the same problem idea (authored by Onufry Wojtaszczyk, Robert Obryk and Adam Polak) was used by us in the Algorithmic Engagements contest in 2014.

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

Problem E. load_balance

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 35 points *10 minute timeout*	The contest is finished.

Problem

At your flat, you take turns in bringing the groceries in. Each day, one person goes out and does the shopping for everybody who lives in the whole building. Today it's your turn, you did the shopping, and all these bags are heavy!
You can't do much about the fact the bags are heavy — people depend on you to bring the bags home. But they will be easier to carry if you distribute the load equally between the left and the right hand. So, you look at all the bags you have, and wonder whether it's possible to split them so that the weight of the ones you'll carry in your left hand will be equal to the weight of those you will carry in your right hand.

Input

The input library will be called "load_balance", see the sample inputs below for examples in your language. It will define two methods: GetN(), which will return the number of bags you have to carry, and GetWeight(i), which will return the weight of the ith bag, for $0 \le i < N$.

Output

Output one string: "IMPOSSIBLE" if it is impossible to split the load equally, or "POSSIBLE" if it is possible (quotes are for clarity only).

Limits

Each node will have access to 512MB of RAM, and a time limit of 4 seconds.
$1 \le \text{GetWeight(i)} \le 10^{15}$ for all i with $0 \le i < \text{GetN()}$.

Small input

Your solution will run on 10 nodes.
$1 \le \text{GetN()} \le 30$.

Large input

Your solution will run on 100 nodes.
$1 \le \text{GetN()} \le 52$.

Sample

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

Note: this problem might be known to a few people, since we communicated it externally when speaking about Distributed Code Jam.

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

Powered by

Google Cloud Platform

Distributed Online Round

Submissions

Testrun
0pt | Not attempted **0/64 users** correct (0%)

almost_sorted
1pt | Not attempted **194/203 users** correct (96%)
7pt | Not attempted **104/187 users** correct (56%)

mutexes
2pt | Not attempted **84/147 users** correct (57%)
20pt | Not attempted **48/69 users** correct (70%)

johnny
2pt | Not attempted **91/105 users** correct (87%)
30pt | Not attempted **17/70 users** correct (24%)

highest_mountain
1pt | Not attempted **43/61 users** correct (70%)
37pt | Not attempted **0/9 users** correct (0%)

Top Scores

mk.al13n	63
ecnerwala	63
shik	63
Marcin.Smulewicz	56
WJMZBMR	56
bmerry	43
ZbanIlya	43
wan92hy	42
simonlindholm	42
dreamoon	42

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. almost_sorted

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 7 points *10 minute timeout*	The contest is finished.

Don't know what distributed problems are about? See our guide.

Problem

As a *very important director* of a *very important company*, you have a lot of files to keep track of. You do this by carefully keeping them sorted. However, you took a well-deserved vacation last week, and when you came back, you discovered to your horror that someone has put the files out of place!

They are not very much out of place, in fact they're still *almost* sorted. More precisely, the file that should be in position i if the files were sorted is now at most K positions away — that is, somewhere between position $i - K$ and $i + K$, inclusive.
However, you can't work like this. So, you ask your assistants to put the files into their correct places. Each file has an identifier, and files with a larger identifier should be placed after those with the smaller identifier. They will not change the relative order of files with the same identifier.
To verify the files are sorted correctly, you will ask your assistants to calculate a simple checksum — for each file multiply that file's identifier by its position (beginning from 0), and sum this for all files, modulo 2^{20}.

Unfortunately, to make use of the checksum, you have to know what it's value should be. So, write a program that will output the expected checksum after sorting the files.

Input

The input library will be called "almost_sorted"; see the sample inputs below for examples in your language. It will define three methods: NumberOfFiles(), which will return the number of files, MaxDistance() — the maximum difference between the current and desired position of any file, and Identifier(i), which will return the value of the identifier of the file that's currently standing on position i, for $0 \le i <$ NumberOfFiles().
A single call to Identifier() will take approximately 0.04 microseconds.

Output

Output one number — the value of the checksum for the sorted sequence of files, modulo 2^{20}.

Limits

Each node will have access to 128MB of RAM, and a time limit of 3 seconds.
$0 \le$ Identifier(i) $\le 10^{18}$, for $0 \le i <$ NumberOfFiles().

Small input

Your solution will run on 10 nodes.
$0 \le$ MaxDistance() $<$ NumberOfFiles() ≤ 1000.

Large input

Your solution will run on 100 nodes.
$1 \le$ NumberOfFiles() $\le 10^8$.
$0 \le$ MaxDistance() $<$ NumberOfFiles().
$0 \le$ MaxDistance() $\le 10^6$.

Sample

Input	Output
See sample input files below.	For sample input 1: 0 For sample input 2: 7000 For sample input 3: 154112

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

Problem C. mutexes

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 20 points *10 minute timeout*	The contest is finished.

Don't know what distributed problems are about? See our guide.

Problem

In writing multi-threaded programs one of the big problems is to prevent concurrent access to data. One of the more common mechanisms for doing this is using *mutual exclusion locks* (also called *mutexes*). A *mutex* is something that can be acquired by only one thread at a time. If one thread has already acquired the mutex, and a second thread tries to acquire it, the second thread will wait until the first thread releases the mutex, and only then will it proceed (with acquiring the mutex and doing whatever it planned on doing next).

A danger when using mutexes is *deadlock* — a situation where some threads block each other and will never proceed. A deadlock occurs when each one thread has already acquired mutex **A**, and now tries to acquire mutex **B**, while another thread has already acquired mutex **B** and tries to acquire mutex **A** (more complex scenarios with more threads are also possible, but we will only be concerned with the two-thread situation).

You are now analyzing a two-threaded program, and trying to determine whether it will deadlock. You know the exact set of mutex operations (acquire and release) each of the two threads will perform, in order. However, you do not know how quickly each thread will perform each of its operations — it is possible, for instance, for one thread to perform almost all of its operations, then for the other thread to catch up, and then for the first thread to proceed.

You are interested in determining whether it is possible that the two threads will deadlock. Initially all the mutexes are released. We assume that when a thread has finished all of its operations, it releases all the mutexes it still has acquired.

Input

The input library will be called "mutexes"; see the sample inputs below for examples in your language. It will define two methods: NumberOfOperations(i), which will return the number of operations thread *i* performs (*i* has to be 0 or 1), and GetOperation(i, index), which will report what the *index*th operation performed by thread *i* is (where *i* is 0 or 1, and $0 \le index <$ NumberOfOperations(i)). This will be a positive number *X* if the *index*th operation is to acquire mutex *X*, and a negative number *-X* if the *index*th operation is to release mutex *X*.
The sequence of operations for a single thread will always be valid, that is, a given thread will never try to acquire a lock it has already acquired (and not yet released), or release a lock it has already released (and not yet acquired) or has never acquired in the first place. A thread's first operation on a lock (if any) will always be an acquire operation.
One call to GetOperation will take approximately 0.005 microseconds, with the exception of the first call, which will cache the input values and might take up to 100 milliseconds.

Output

Output the smallest total number of operations the two threads can perform before deadlocking (including the last two acquire operations), if a deadlock is possible, or the word 0K if a deadlock can't happen.

Limits

Each node will have access to 256MB of RAM, and a time limit of 4 seconds.
$-10^5 \le$ GetOperation(i, index) $\le 10^5$ for all valid *i* and *index*. GetOperation will never return 0.

Small input

Your solution will run on 10 nodes.
$1 \le$ NumberOfOperations(i) ≤ 1000 for both possible values of *i*.

Large input

Your solution will run on 100 nodes.
$1 \le \text{NumberOfOperations}(i) \le 4 \times 10^4$ for both possible values of *i*.

Sample

Input	Output
See sample input files below.	For sample input 1: OK For sample input 2: 7 For sample input 3: 6

The fastest way to deadlock in the third example is for the first thread to perform the first three operations (ending up with mutexes 1, 2 and 3), then for the second thread to perform the first operation (acquiring mutex 4). At this point both threads try to perform one operation more (the first thread trying to acquire mutex 4, the second thread trying to acquire mutex 3) and deadlock.

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

Problem D. johnny

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 30 points *10 minute timeout*	The contest is finished.

Don't know what distributed problems are about? See our guide.

Problem

You and Johnny play a very simple card game. Both players have a deck of cards. Both draw a card at random from their deck, and the player with the better card wins. "Better" is a complex concept, but for any two cards, you (and Johnny) know which one is better, with no ties allowed. Note that this is not necessarily transitive: if card A is better than B, and B is better than C, it is possible that C is better than A.

Johnny is very, very unhappy when he loses. So, you would like to make sure that he will win. You have N cards, and you want to split them into two non-empty decks (with no cards left over), one for you and one for Johnny, so that whatever card you draw from your deck and whatever card Johnny draws from his, Johnny will win.

If it is possible to do this in multiple ways, you want Johnny's deck to be as large as possible (as long as your deck is not empty).

Input

The input library will be called "johnny"; see the sample inputs below for examples in your language. It will define two methods: NumberOfCards(), which will return the number of cards, and IsBetter(i, j), which will return true if card i is better than card j, for $0 \le i, j <$ NumberOfCards(). If called with $i=j$, it will return false. One call to IsBetter will take approximately 0.03 microseconds.

Output

If it is possible to split the cards so that Johnny will always win, output one number: the largest possible size of Johnny's deck. If it is impossible to split the cards in such a way, output the word `IMPOSSIBLE`.

Limits

Each node will have access to 256MB of RAM, and a time limit of 3 seconds.

Small input

Your solution will run on 10 nodes.
$2 \le$ NumberOfCards() ≤ 1000.

Large input

Your solution will run on 100 nodes.
$2 \le$ NumberOfCards() $\le 2 \times 10^4$.

Sample

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

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

Problem E. highest_mountain

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.

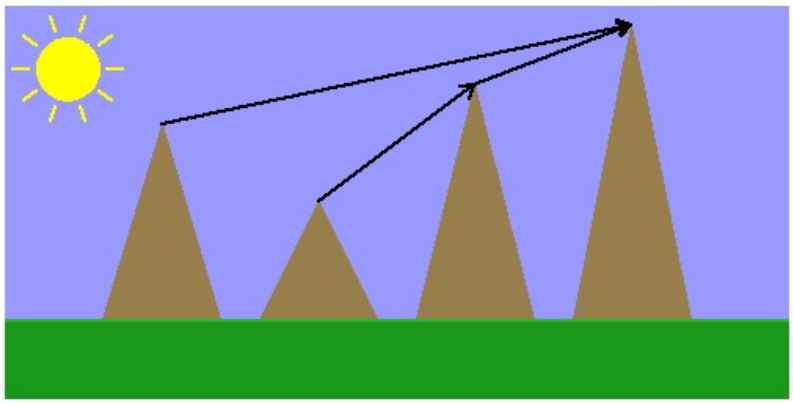
small 1 points *2 minute timeout*	The contest is finished.
large 37 points *10 minute timeout*	The contest is finished.

Don't know what distributed problems are about? See our guide.

Problem

You were born and live in a small town in a remote mountain range extending east to west. In this mountain range there is a peak every kilometer, and there are no intermediate peaks. Recently, you checked on the Internet what the highest peak in the range is, and were suprised — from your town a different peak seems to be the highest. And when you went for a walk to the nearest mountain top, yet another peak appeared to be highest. You're not sure the Internet data is correct (they write all sorts of stuff on the Internet!), so you decided to compile your own list of potentially highest peaks in the range.

Your list will contain every peak **B** with the following property: if **B** is visible from some other peak **A**, no peak beyond **B** is visible from **A**. Formally, this means that if **B** lies, say, to the east of **A**, then all peaks between **A** and **B** are below the line connecting **A** and **B**, and all the peaks to the east of **B** are below or on that line.



In this example, the fourth peak is the last one you see to the east from the first and third peaks. From the second peak, to the east you only see the third peak. Your list will include only peaks 1 and 4: peak 2 is visible from peak 3, but is not the farthest (1 is), and peak 3 is visible from peak 1, but is not the farthest (4 is).

The rationale for this criterion is that you figure that if from some peak **A** you can see peak **C**, and you can also see some other peak **D** that lies in the same direction and is more distant, then **C** is'obviously not the highest peak in the range (because either **A**, or **D**, is higher). You don't trust your intuition any more, so even if the highest visible peak in any direction appears to be much lower than the one you're standing on (for instance, you are standing on a peak of height 1000, and the next and last peak to the east is of height 1), you will consider the peak of height 1 to be a candidate for your list.

Input

The input library will be called "highest_mountain"; see the sample inputs below for examples in your language. It will define two methods:
NumberOfPeaks(), which will return the number of peaks in the range, and
GetHeight(i), which will return the height of the *i*th peak from the west, for $0 \le i < \text{NumberOfPeaks}()$.
One call to GetHeight will take approximately 0.1 microseconds.

Output

Output one number: the total number of the peaks that will be included in your list.

Limits

Each node will have access to 128MB of RAM, and a time limit of 6 seconds.
$0 \le \text{GetHeight}(i) \le 10^9$ for all i with $0 \le i < \text{NumberOfPeaks}()$.

Small input

Your solution will run on 10 nodes.
1 ≤ NumberOfPeaks() ≤ 1000.

Large input

Your solution will run on 100 nodes.
1 ≤ NumberOfPeaks() ≤ 4×10^8.

Sample

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

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

Distributed Finals 2015 Warmup

Submissions

Testrun

0pt	Not attempted **0/5 users** correct (0%)

majority

1pt	Not attempted **10/10 users** correct (100%)
20pt	Not attempted **8/8 users** correct (100%)

Top Scores

simonlindholm	21
mk.al13n	21
bmerry	21
wan92hy	21
WJMZBMR	21
shik	21
ZbanIlya	21
Marcin.Smulewicz	21
dreamoon	1
MiSawa	1

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. majority

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 20 points *10 minute timeout*	The contest is finished.

Problem

Your country is electing its president, and you are in charge of the new electronic voting system. The citizens have voted, and now you have to check if any of the candidates obtained a *majority* — that is, if there is a candidate for whom more than half of the citizens voted.

Input

The input library will be called "majority", see the sample inputs below for examples in your language. It will define two methods: GetN(), which will return the number of voting citizens *N*, and GetVote(i), which will (for $0 \le i < N$) return the identifier of the candidate for whom citizen *i* voted.

Output

If any candidate obtained a majority of the votes, output the identifier of that candidate. Otherwise, output the string "NO WINNER" (quotes for clarity only). A single call to GetVote(i) will take approximately 0.025 microseconds.

Limits

Each node will have access to 128MB of RAM, and a time limit of 3 seconds.
$0 \le$ GetVote(i) $\le 10^9$ for all *i* with $0 \le i < N$.

Small input

Your solution will run on 10 nodes.
$1 \le$ GetN() ≤ 1000.

Large input

Your solution will run on 100 nodes.
$1 \le$ GetN() $\le 10^9$.

Sample

Input	Output
See the input files below.	For sample input 1: 7 For sample input 2: NO WINNER For sample input 3: NO WINNER

Note: the same problem idea was used by us in a tutorial in the Algorithmic Engagements contest in 2014.

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


Powered by
Google Cloud Platform

Distributed Finals 2015

Submissions

Testrun
0pt | Not attempted **0/6 users** correct (0%)

kolakoski
8pt | Not attempted **5/7 users** correct (71%)
17pt | Not attempted **2/5 users** correct (40%)

necklace
16pt | Not attempted **10/10 users** correct (100%)
29pt | Not attempted **9/10 users** correct (90%)

rocks
7pt | Not attempted **2/2 users** correct (100%)
53pt | Not attempted **0/1 users** correct (0%)

shipping
26pt | Not attempted **2/6 users** correct (33%)
44pt | Not attempted **0/1 users** correct (0%)

Top Scores

bmerry	103
Marcin.Smulewicz	71
shik	70
MiSawa	60
ZbanIlya	53
WJMZBMR	45
simonlindholm	45
mk.al13n	45
wan92hy	45
dreamoon	24

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.

Distributed Finals 2015

A. Testrun
B. kolakoski
C. necklace
D. rocks
E. shipping

Contest Analysis
Questions asked 2

Submissions

Testrun
0pt | Not attempted
0/6 users correct (0%)

kolakoski
8pt | Not attempted
5/7 users correct (71%)
17pt | Not attempted
2/5 users correct (40%)

necklace
16pt | Not attempted
10/10 users correct (100%)
29pt | Not attempted
9/10 users correct (90%)

rocks
7pt | Not attempted
2/2 users correct (100%)
53pt | Not attempted
0/1 users correct (0%)

shipping
26pt | Not attempted
2/6 users correct (33%)
44pt | Not attempted
0/1 users correct (0%)

Top Scores

bmerry	103
Marcin.Smulewicz	71
shik	70
MiSawa	60
ZbanIlya	53
WJMZBMR	45
simonlindholm	45
mk.al13n	45
wan92hy	45
dreamoon	24

Problem B. kolakoski

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

Problem

The Kolakoski sequence is defined as follows, where A(i) is the i-th term in the sequence:

- A(0) = 1
- A(1) = 2
- The sequence is composed entirely of alternating runs of 1's and 2's
- A(i) is the length of the i-th run.

This completely and uniquely defines the sequence.

The first twenty terms of the sequence are as follows, where the lines mark the alternating runs of 1's and 2's:

1 2 2 1 1 2 1 2 2 1 2 2 1 1 2 1 1 2 2 1

1 2 2 1 1 2 1 2 2 1 2 1 1 2 1 2 2 1

By collecting the lengths of each run, we obtain the same sequence again.

You become mystified contemplating the elegance of the Kolakoski sequence and after staring at its 1's and 2's for far too long, you begin to wonder if maybe you should spice it up a little and introduce some more numerical variety to the terms.

So you decide to assign an arbitrary coefficient to each index in a manner such as the following:

C(0)=1
C(1)=3
C(2)=1
C(3)=5
C(4)=2
C(5)=2

By multiplying the first 6 terms each by their coefficient and summing, we get

1*1 + 3*2 + 1*2 + 5*1 + 2*1 + 2*2 = 20.

Given a mapping from index to coefficient, find the dot product of the first **N** terms of the Kolakoski sequence and their respective coefficients.

Input

The library "kolakoski" will contain two functions:

- GetIndex() which returns **N**, the number of terms we wish to sum; and
- GetMultiplier(i) which takes an index i and returns the coefficient (a number from 0 to 50) for that index.

A single call to GetMultiplier will take approximately 0.005 microseconds.

Output

Output one number: the weighted sum of the elements of the Kolakoski sequence.

Limits

Each node will have access to 700MB of RAM.
Your solution will run on 100 nodes in both inputs.

Small input

GetMultiplier(i) will always return 1, for all the inputs.
$1 \le \text{GetIndex}() \le 10^9$
Each node will have a time limit of 10 seconds.

Large input

$1 \le$ GetMultiplier(i) ≤ 50 for all i
$1 \le$ GetIndex() $\le 3 \times 10^9$
Each node will have a time limit of 12 seconds.

Sample

Input	Output
See below for sample input files.	For sample input 1: 1 For sample input 2: 15 For sample input 3: 50

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

Problem C. necklace

small 16 points *2 minute timeout*	The contest is finished.
large 29 points *10 minute timeout*	The contest is finished.

Problem

You've come up with the coolest idea ever for a new fashion trend: customizable necklaces made out of strings with beads that display letters and other characters! The beads appear only on the front of the necklace and read only in one direction, so the string of characters is not circular and irreversible. By itself, this is not really a new idea. The awesome new feature you have in mind is to add a button that lights up some of the beads so that they display a secret message consisting of characters that form a subsequence of the main string of characters. This will have so many applications... just think of the possibilities! And it's so shiny! People are going to love it! Everyone will want their own!

So you announce this product, allowing people to place orders for necklaces by specifying the string of characters to be displayed on the necklace as well as the secret message to be lit up when they press the button. The orders come pouring in! Your idea is even more popular than you expected! How exciting!

Unfortunately, after examining a few orders, you realize that you forgot to check the crucial constraint that the secret message has to be a subsequence of the main necklace string. Without that, the secret message can't always be lit up entirely.

You don't want to disappoint your customers by just telling them that it is impossible to light up their secret messages in the chosen necklace strings. So you decide to offer them an alternative message by finding a substring of their secret message that forms a subsequence of their necklace string, in case they would be satisfied with this shorter version. You want to maximize the length of such a substring.

Given a necklace string **N** and a secret message string **M**, find the maximum length of a substring of **M** that is also a subsequence of **N**.

Input

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

- GetNecklaceLength(), which returns the length of the necklace string
- GetNecklaceElement(i), which returns the i-th (0-indexed) element of necklace string
- GetMessageLength(), which returns the length of the secret message
- GetMessageElement(i), which returns the i-th (0-indexed) element of the secret message.

A single call of GetNecklaceElement or GetMessageElement will take up to 0.02 microseconds.

Output

Output one integer - the maximum length of a substring of Message that is also a subsequence of Necklace.

Limits

$0 \le$ GetNecklaceElement(i), GetMessageElement(i) $\le 10{,}000$
$1 \le$ GetNecklaceLength() $\le 10^9$
Each node will have access to 256MB of RAM and a time limit of 5 seconds.
Your solution will run on 100 nodes (both for the small and the large input).

Small input

$1 \le$ GetMessageLength() ≤ 100

Large input

$1 \le$ GetMessageLength() ≤ 3000

Sample

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

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

Problem D. rocks

small 7 points *2 minute timeout*	The contest is finished.
large 53 points *10 minute timeout*	The contest is finished.

Problem

You own an almond farm in a region which for the last few years has been experiencing an extreme, record-breaking drought. Reservoirs are drying up, mandatory water rationing is in effect, and you are facing pressure to do something about this farm of yours that is consuming 4 liters of water per almond.

As you survey the regular grid-like arrangement of your farm, a brilliant idea strikes you. You should turn the land into a giant board where people can play life-sized versions of their favorite grid-based board games! So you take out all the almond trees and turn your farm into an amusement park by sectioning your land into a square **N**-by-**N** grid.

Unfortunately, the region you are in is also known for its earthquakes, and soon after you convert your land, an earthquake strikes, triggering a landslide that drops giant rocks onto your grid, each rock conveniently occupying the space of exactly one of the cells. No cell contains more than one rock.

You are now trapped in the bottom left corner (the cell with coordinates (0, 0)), and you need to get to the exit at the top right corner (the cell with coordinates (**N**-1, **N**-1)). You can move only up or to the right 1 cell at a time and cannot squeeze past any rocks or climb over them, so if you want to move into a cell that is occupied by a rock, you have to push it into next cell in the same direction as you are moving. If there is already a rock in that cell, it will also get pushed in the same direction, and so on, until finally there is a square without rocks. After every push, each rock occupies exactly one cell. You can push up to **K** rocks at the same time in this manner, but you cannot push rocks off the grid. Is it possible to reach the exit?

Input

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

- GetN(), which returns the number of rows of the grid (which is equal to the number of columns);
- GetK(), which returns the maximum number of rocks you can push in front of you; and
- IsRock(x, y), which returns true if there is a rock in the cell with coordinates (x, y).

A single call to IsRock() will take approximately 0.05 microseconds.

Output

Output one line containing the word "YES" if it is possible to reach the exit, or the word "NO" if it is impossible.

Limits

Each node will have access to 600MB of RAM, and a time limit of 4 seconds.
IsRock(0, 0) and IsRock(GetN() - 1, GetN() - 1) will return false.
$0 \le$ GetK() $\le$ GetN()
Your solution will run on 100 nodes in both inputs.

Small input

$2 \le$ GetN() $\le 2{,}000$

Large input

$2 \le$ GetN() $\le 10{,}000$

Sample

Input	Output
See input files below.	For sample input 1: YES

```
                        For sample input 2:
                        NO
                        For sample input 3:
                        NO
```

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

Problem E. shipping

small 26 points *2 minute timeout*	The contest is finished.
large 44 points *10 minute timeout*	The contest is finished.

Problem

In a certain country that shall remain unnamed, people live in **N** villages along river valleys separated by mountain ranges. Due to the geography of the country, it is extremely difficult to cross over mountains. The only practical way to get from one village to another is by following rivers. All of the rivers are part of the same system and each river connects to another river downstream, until eventually all water in the system passes through a single point to enter the sea. There is a village at every point where two rivers join and at each river's source, and there is one village at the mouth of the entire river system, where it meets the sea. There may be also be villages at other points along rivers. No two villages are at the same location along a river, and there is exactly one path between any two villages. In other words, the villages and the paths between them form a tree.

You are in charge of managing a shipping company that delivers packages between villages by transporting them on boats along the rivers. Since there is only one path between any two villages, you fortunately do not have to worry about finding the shortest path to use. Unfortunately, however, the country is in the middle of a civil war, with **K** rival factions battling for control. Each village is under the control of exactly one of the factions. Luckily, conditions are stable at the moment and no village is going to change factions anytime soon.

As a neutral company, you are able to send your boats through every village, but only under the condition that each of the factions is allowed to use your shipping services for free, by loading additional packages onto any of your boats that pass between villages under the control of that faction. Every time one of your boats passes through a village (including the village at which your boat begins its route), you may be given an additional package occupying 1 unit of capacity on your boat, to be transported to another village further along on your boat's current route that is occupied by the same faction. Because the factions do not want their packages to be intercepted by other factions, you will be given a package to transport between two villages only if every village on the path between them (including the destination) is occupied by the same faction. Once you deliver a package, you can reuse the space for another package later on, but multiple packages at the same time require multiple units of capacity.

You are now faced with the problem of guaranteeing enough extra capacity on your boats to transport these extra packages. You have **Q** shipments, each of which has a source village and a destination village. You will use a different boat for each shipment. For each shipment, determine the number of units of capacity to reserve on your boat in order to carry all of the additional packages for the various factions in the worst case.

Input

The input library is called "shipping"; see the sample inputs below for examples in your language. It defines 6 methods:

- NumberOfVillages()
- VillageFaction(village_index)
- VillageImmediatelyDownstream(village_index)
- NumberOfShipments()
- GetShipmentSource(shipment_id)
- GetShipmentDestination(shipment_id)

Villages and shipments are both zero-indexed. VillageImmediatelyDownstream returns the index of the village that lies immediately downstream, except for the village at the mouth of the river system, whose return value is the index of this village (since no other village is downstream, just the sea).
A single call of VillageImmediatelyDownstream will take approximately 0.04 microseconds. A single call of VillageFaction will take approximately 0.02 microseconds.

Output

Output a space-separated list of integers, where the i-th integer is the minimum number of units of capacity needed for the boat delivering the i-th shipment to carry all additional packages in the worst case.

Limits

Each node will have access to 256MB of RAM, and a time limit of 5 seconds.
$1 \le$ NumberOfVillages() $\le 10^8$
$0 \le$ VillageFaction(i) $\le 10^6$ for all villages
$0 \le$ GetShipmentSource(i), GetShipmentDestination(i) $<$ NumberOfVillages() for all shipments
The VillageImmediatelyDownstream method will describe a tree (that is, there will be only one path between any two villages).
Your solution will run on 100 nodes in both inputs.

Small Input

$1 \le$ NumberOfShipments() ≤ 10
VillageImmediatelyDownstream(i) $\le$ i for all villages

Large input

$1 \le$ NumberOfShipments() $\le 20{,}000$

Sample

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

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