Distributed Round 2 2016

Submissions

Testrun
0pt | Not attempted **0/74 users** correct (0%)

again
1pt | Not attempted **401/409 users** correct (98%)
14pt | Not attempted **368/399 users** correct (92%)

lisp_plus_plus
3pt | Not attempted **390/399 users** correct (98%)
17pt | Not attempted **355/385 users** correct (92%)

asteroids
5pt | Not attempted **283/305 users** correct (93%)
25pt | Not attempted **91/170 users** correct (54%)

gas_stations
8pt | Not attempted **191/233 users** correct (82%)
27pt | Not attempted **28/95 users** correct (29%)

Top Scores

eatmore	100
Marcin.Smulewicz	100
tozangezan	100
Errichto.rekt	100
mnbvmar	100
qwerty787788	100
sevenkplus	100
tczajka	100
fhlasek	100
wata	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.

Submissions

Testrun
0pt | Not attempted 0/74 users correct (0%)

again
1pt | Not attempted 401/409 users correct (98%)
14pt | Not attempted 368/399 users correct (92%)

lisp_plus_plus
3pt | Not attempted 390/399 users correct (98%)
17pt | Not attempted 355/385 users correct (92%)

asteroids
5pt | Not attempted 283/305 users correct (93%)
25pt | Not attempted 91/170 users correct (54%)

gas_stations
8pt | Not attempted 191/233 users correct (82%)
27pt | Not attempted 28/95 users correct (29%)

Top Scores

eatmore	100
Marcin.Smulewicz	100
tozangezan	100
Errichto.rekt	100
mnbvmar	100
qwerty787788	100
sevenkplus	100
tczajka	100
fhlasek	100
wata	100

Problem B. again

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

Problem

... we did it again

You know the issue: just as in the Oops problem from Distributed Round 1, we have lost our problem statement and the correct solutions, and we only have these two correct but slow solutions, one per supported language. Once again, we still have our test data. Can you still solve the problem?

Notice that in this problem 20 nodes are used to run both the Small and the Large datasets, which is not the usual number for Distributed Code Jam problems. 20 nodes were also used to run the solutions and produce the answers for the examples.

The C++ solution:

```
#include <message.h>
#include <stdio.h>
#include "again.h"

#define MASTER_NODE 0
#define SENDING_DONE -1
#define LARGE_PRIME 1000000007

int main() {
  if (MyNodeId() == MASTER_NODE) {
    long long result = 0;
    for (int node = 1; node < NumberOfNodes(); ++node) {
      while (true) {
        Receive(node);
        long long value = GetLL(node);
        if (value == SENDING_DONE) {
          break;
        } else {
          result = (result + value) % LARGE_PRIME;
        }
      }
    }
    printf("%lld\n", result);
    return 0;
  } else {
    for (long long i = 0; i < GetN(); ++i) {
      for (long long j = 0; j < GetN(); ++j) {
        long long value = GetA(i) * GetB(j);
        if ((i + j) % NumberOfNodes() == MyNodeId()) {
          PutLL(MASTER_NODE, value);
          Send(MASTER_NODE);
        }
      }
    }
    PutLL(MASTER_NODE, SENDING_DONE);
    Send(MASTER_NODE);
  }
  return 0;
}
```

The Java solution:

```
public class Main {
  static int MASTER_NODE = 0;
  static long SENDING_DONE = -1;
  static long LARGE_PRIME = 1000000007;

  public static void main(String[] args) {
    if (message.MyNodeId() == MASTER_NODE) {
      long result = 0;
      for (int node = 1; node < message.NumberOfNodes(); ++node
        while (true) {
          message.Receive(node);
          long value = message.GetLL(node);
          if (value == SENDING_DONE) {
```

```
            break;
          } else {
            result = (result + value) % LARGE_PRIME;
          }
        }
      }
      System.out.println(result);
    } else {
      for (long i = 0; i < again.GetN(); ++i) {
        for (long j = 0; j < again.GetN(); ++j) {
          long value = again.GetA(i) * again.GetB(j);
          if ((i + j) % message.NumberOfNodes() == message.MyNo
            message.PutLL(MASTER_NODE, value);
            message.Send(MASTER_NODE);
          }
        }
      }
      message.PutLL(MASTER_NODE, SENDING_DONE);
      message.Send(MASTER_NODE);
    }
  }
}
```

Input

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

- **GetN()**:
  - Takes no argument.
  - Returns a 64-bit number.
  - Expect each call to take 0.05 microseconds.
- **GetA(i)**:
  - Takes a 64-bit number in the range 0 ≤ i < GetN().
  - Returns a 64-bit number.
  - Expect each call to take 0.05 microseconds.
- **GetB(i)**:
  - Takes a 64-bit number in the range 0 ≤ i < GetN().
  - Returns a 64-bit number.
  - Expect each call to take 0.05 microseconds.

Output

Output what either of the solutions above would output, if they ran on 20 nodes without any limits on memory, time, number of messages or total size of messages.

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.
Number of nodes: 20.
$0 \le GetA(i) \le 10^9$, for all i.
$0 \le GetB(i) \le 10^9$, for all i.

Small input

$1 \le GetN() \le 30{,}000$.

Large input

$1 \le GetN() \le 10^8$.

Sample

Input	Output
See input files below.	For sample input 1: 999979007 For sample input 2: 2390 For sample input 3: 0

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

Problem C. lisp_plus_plus

small 3 points *2 minute timeout*	The contest is finished.
large 17 points *10 minute timeout*	The contest is finished.

Problem

Lisp++

Alyssa is a huge fan of the Lisp programming language, but she wants it to have even more parentheses, so she is designing a new language, Lisp++. A valid Lisp++ program consists of one of the following. (In this specification, *P* stands for some valid program -- not necessarily the same program each time.)

- `()` Literally, just an opening parenthesis and a closing parenthesis.
- `(`*P*`)` A program within a pair of enclosing parentheses.
- *PP* Two programs (not necessarily the same), back to back.

Alyssa is working on a compiler for Lisp++. The compiler must be able to evaluate a string consisting of `(` characters and/or `)` characters to determine whether it is a valid Lisp++ program, and provide the user with some helpful information if it is not. If the program is valid, the compiler should print -1. Otherwise, it should print the length of the longest prefix that could be extended into a valid program by adding zero or more additional characters to the end. If that prefix is the empty prefix, the compiler should print 0. In particular, if the input string is not a valid program, but can be extended to a valid program, the compiler should print the length of the input string.

For example:

- `()(()())` is a valid program, so the compiler should print -1.
- `((()))` is not a valid program. The prefix `(())` is the longest prefix that is or could be extended into a valid program (in this case, it already is one), so the compiler should print 4. The only longer prefix is `((()))` (i.e., the entire string), but there is no way to add (any number of) characters to the end of that string to make it into a valid program.
- `)` is not a valid program. The prefix `)` cannot be extended into a valid program. The empty prefix is not a valid program, but it can easily be extended into one (by adding `()`, for example). So the compiler should print 0.

Given a string, what should Alyssa's compiler print?

Input

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

- **GetLength()**:
  - Takes no argument.
  - Returns a 64-bit integer: the length of the string.
  - Expect each call to take 0.07 microseconds.
- **GetCharacter(i)**:
  - Takes a 64-bit integer in the range $0 \le i <$ GetLength().
  - Returns a character: either `(` or `)`, the character at position i.
  - Expect each call to take 0.07 microseconds.

Output

Output a single line with a single integer: what the compiler should print, as described above.

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.
GetCharacter(i) is always `(` or `)`.

Small input

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

Large input

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

Sample

Input	Output
See input files below.	For sample input 1: -1 For sample input 2: 4 For sample input 3: 0

The sample input files are the same examples from the problem statement, in the same order.

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

Problem D. asteroids

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

Problem

Asteroids

You are playing a game of DCJAsteroids in your old arcade machine. The game is played on a screen that displays a matrix of square cells. The matrix has a fixed width and infinite height. Each cell may contain your spaceship, an asteroid, or some amount of helium. The goal of the game is to collect helium without colliding with any asteroids. Your spaceship always stays in the bottom row of the screen, and it can only move horizontally.

The game is turn-based. On a turn, you can remain where you are, move to the cell on your left (unless you are in the left-most column), or move to the cell on your right (unless you are in the right-most column). After you move (or not) your spaceship, every other item in the screen moves to the cell directly below it. The cells in the bottom row fall off the bottom of the screen and disappear.

For instance, suppose this is a current arrangement, with X representing your spaceship, # representing a cell occupied by an asteroid, and . representing a cell not occupied by an asteroid (to keep it simple, we do not depict the amount of helium in this example).

```
#.##.
#..#.
..X..
```

If you decide to stay at your current position, then the resulting state after the turn is resolved will be:

```
#.##.
#.X#.
```

If you decide to move left, then the result after the turn is resolved will be:

```
#.##.
#X.#.
```

If you ever move the spaceship into a cell occupied by an asteroid or an asteroid moves into the cell occupied by your spaceship, the spaceship is destroyed. In the example above, if you decided to move right, after the turn is resolved, your spaceship and an asteroid would occupy the same cell, so the spaceship would be destroyed. If you decided to move left two turns in a row, the second move would make your spaceship move into an asteroid, so it would be destroyed as well.

Each cell not occupied by an asteroid may have a different amount of helium. We assign a digit 0 through 9 to each non-asteroid cell (instead of .); the number in a cell represents the number of points you earn from the helium in that cell. Each time your spaceship moves into a cell containing helium or a cell containing helium moves into your spaceship's position, you collect the helium there and are awarded the points. You can only collect the helium in each cell once; after that, your spaceship can still be in that cell, but you collect no additional helium and receive no additional points for it.

You will be given a matrix of characters representing the asteroids and helium that are approaching. You may assume that all rows below and above the input matrix are full of worthless helium (i.e., they are rows of 0s). You may choose the position you are in when the bottom row in the input matrix falls into your current row.

If there is no way to navigate through the entire input matrix without the spaceship being destroyed, output -1. Otherwise, output the maximum number of points you can accumulate while navigating through the entire input matrix. You are considered done navigating only when every row in the input matrix has already disappeared from the screen.

Let us add point values to the empty cells in the example above (assume your current position has 0 value). If you start by going left and then stay for two turns, you would go through these states and emerge without being destroyed:

```
#1##9    00000    00000     00000
#23#9 -> #1##9 -> 00000  -> 00000
66X15    #X3#9    #X##9     0X000
```

In this case you collect 6+2 points in the first turn, 1 in the second turn and 0 in the third, yielding a total of 9 points. If instead you stay for the first turn, then move left, and then stay again, this would happen:

```
#1##9    00000    00000     00000
#23#9 -> #1##9 -> 00000  -> 00000
66X15    #2X#9    #X##9     0X000
```

In this case you collect 3 points in the first turn, 2+1 points in the second and 0 in the third, for a total of 6 points. Any other sequence of moves gets the spaceship destroyed.

Input

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

- **GetHeight()**:
  - Takes no argument.
  - Returns a 64-bit integer: the height of the input matrix.
  - Expect each call to take 0.05 microseconds.
- **GetWidth()**:
  - Takes no argument.
  - Returns a 64-bit integer: the width of both the screen and the input matrix.
  - Expect each call to take 0.05 microseconds.
- **GetPosition(i, j)**:
  - Takes two 64-bit integers in the ranges $0 \le i <$ GetHeight(), $0 \le j <$ GetWidth().
  - Returns a character: the contents of the cell at row i and column j of the input matrix. The character is # if the cell contains an asteroid and a digit if not, representing the point value of the helium you can collect from that cell, as explained above. Rows are numbered from bottom to top (so your spaceship will enter the rows in increasing order). Columns are numbered from left to right, as explained above.
  - Expect each call to take 0.05 microseconds.

Output

Output a single line with a single integer: the maximum number of points you can accumulate in your journey through the asteroids without being destroyed, or -1 if it is impossible to avoid destruction.

Limits

Time limit: 5 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.
GetPosition(i, j) is a digit or #, for all i and j.

Small dataset

Number of nodes: 10.
$1 \le$ GetHeight() ≤ 1000.
$1 \le$ GetWidth() ≤ 1000.

Large dataset

Number of nodes: 100.
$1 \le$ GetHeight() $\le 30{,}000$.
$1 \le$ GetWidth() $\le 30{,}000$.

Sample

Input	Output
See input files below.	For sample input 1: 10 For sample input 2: 22 For sample input 3: -1

For ease of reading, these are the input matrices in the samples:

```
8##123
#999#1
21##11
52#11#

1#78
0011
#2#9
0136
0#8#
21#9

0##
000
##0
```

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

Problem E. gas_stations

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

Problem

Gas Stations

You are about to embark on the road trip of a lifetime, covering millions or even billions of kilometers! Of course, gas for such a long trip can get really expensive, and you are on a budget, so you should plan ahead. Fortunately, the road you are taking has a lot of gas stations: there is one at your starting point, one every kilometer thereafter. There is no gas station at your ending point (not that it would do you any good).

Different gas stations may charge different prices per liter of gas. Your car is really old, so it can run for exactly 1 kilometer with 1 liter of gas. You can never have more liters of gas than the tank can hold, but you can add gas to the tank at any station; you do not need to wait for the tank to be empty. Each time you advance a kilometer, your tank's contents are reduced by 1 liter. It is OK if the tank becomes empty exactly at the end of your trip or exactly as you reach a gas station (in which case you can add gas there).

Given the length of your route, the size of your tank and the price per liter of each station, what is the minimum amount of money you need to complete your trip? Your starting point is exactly at the station at the 0 kilometer mark, and your tank starts empty.

Input

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

- **GetNumKms()**:
  - Takes no argument.
  - Returns a 64-bit integer: the total length in kilometers of your trip.
  - Expect each call to take 0.15 microseconds.
- **GetTankSize()**:
  - Takes no argument.
  - Returns a 64-bit integer: the maximum amount of liters of gas you can have in your tank.
  - Expect each call to take 0.15 microseconds.
- **GetGasPrice(i)**:
  - Takes a 64-bit integer in the range $0 \le i <$ GetNumKms().
  - Returns a 64-bit integer: the price of each liter of gas in the station exactly i kilometers from your starting point.
  - Expect each call to take 0.15 microseconds.

Output

Output a single line with a single integer: the minimum amount of money you need to pay for all the gas necessary for your trip.

Limits

Time limit: 5 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.
$1 \le$ GetTankSize() $\le$ GetNumKms().
$1 \le$ GetGasPrice(i) $\le 10^9$, for all i.

Small dataset

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

Large dataset

Number of nodes: 100.
$1 \le$ GetNumKms() $\le 5 \times 10^8$.

Sample

Input	Output
See input files below.	For sample input 1: 7 For sample input 2: 7 For sample input 3: 11

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