Distributed Round 1 2016

A. Testrun
B. oops
C. rps
D. crates
E. winning_move

Contest Analysis

Questions asked 8

Submissions

Testrun
0pt | Not attempted 0/422 users correct (0%)

oops
2pt | Not attempted 893/925 users correct (97%)
12pt | Not attempted 756/882 users correct (86%)

rps
1pt | Not attempted 789/857 users correct (92%)
15pt | Not attempted 585/783 users correct (75%)

crates
8pt | Not attempted 557/673 users correct (83%)
25pt | Not attempted 258/433 users correct (60%)

winning_move
3pt | Not attempted 635/700 users correct (91%)
34pt | Not attempted 49/309 users correct (16%)

Top Scores

simonlindholm	100
tomconerly	100
eatmore	100
cgy4ever	100
bmerry	100
Simon.M	100
Klockan	100
tczajka	100
tkociumaka	100
Zlobober	100

Problem B. oops

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

Problem

Oops

The team preparing the Distributed Code Jam made a mess and needs your help. The statement and solutions for this problem were lost minutes before the contest, and all that we could recover were these two correct but far too slow (and misguided) solutions, one per language. Fortunately, we still have the test data. Can you reconstruct the statement and solve the problem properly based on the recovered slow solutions?

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 "oops.h"

#define MASTER_NODE 7
#define DONE -1

int main() {
  long long N = GetN();
  long long nodes = NumberOfNodes();
  long long my_id = MyNodeId();
  long long best_so_far = 0LL;
  for (long long i = 0; i < N; ++i) {
    for (long long j = 0; j < N; ++j) {
      if (j % nodes == my_id) {
        long long candidate = GetNumber(i) - GetNumber(j);
        if (candidate > best_so_far) {
          best_so_far = candidate;
          PutLL(MASTER_NODE, candidate);
          Send(MASTER_NODE);
        }
      }
    }
  }
  PutLL(MASTER_NODE, DONE);
  Send(MASTER_NODE);

  if (my_id == MASTER_NODE) {
    long long global_best_so_far = 0;
    for (int node = 0; node < nodes; ++node) {
      long long received_candidate = 0;
      while (true) {
        Receive(node);
        received_candidate = GetLL(node);
        if (received_candidate == DONE) {
          break;
        }
        if (received_candidate > global_best_so_far) {
          global_best_so_far = received_candidate;
        }
      }
    }
    printf("%lld\n", global_best_so_far);
  }
  return 0;
}
```

The Java solution:

```
public class Main {
  static int MASTER_NODE = 7;
  static int DONE = -1;
```

```
  public static void main(String[] args) {
    long N = oops.GetN();
    long nodes = message.NumberOfNodes();
    long my_id = message.MyNodeId();
    long best_so_far = 0L;
    for (long i = 0; i < N; ++i) {
      for (long j = 0; j < N; ++j) {
        if (j % nodes == my_id) {
          long candidate = oops.GetNumber(i) - oops.GetNumber(j
          if (candidate > best_so_far) {
            best_so_far = candidate;
            message.PutLL(MASTER_NODE, candidate);
            message.Send(MASTER_NODE);
          }
        }
      }
    }
    message.PutLL(MASTER_NODE, DONE);
    message.Send(MASTER_NODE);

    if (my_id == MASTER_NODE) {
      long global_best_so_far = 0;
      for (int node = 0; node < nodes; ++node) {
        long received_candidate = 0;
        while (true) {
          message.Receive(node);
          received_candidate = message.GetLL(node);
          if (received_candidate == DONE) {
            break;
          }
          if (received_candidate > global_best_so_far) {
            global_best_so_far = received_candidate;
          }
        }
      }
      System.out.println(global_best_so_far);
    }
  }
}
```

Input

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

- **GetN()**:
  - Takes no argument.
  - Returns a 64-bit number.
  - Expect each call to take 0.05 microseconds.
- **GetNumber(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: 6 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.
$-10^{18} \le$ GetNumber(i) $\le 10^{18}$, for all i.

Small dataset

1 ≤ GetN() ≤ 30,000.

Large dataset

1 ≤ GetN() ≤ 10^9.

Sample

Input	Output
See input files below.	For sample input 1: 93 For sample input 2: 0 For sample input 3: 14

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