Code Jam 2018 - Round 3

Name-Preserving Network

Problem

A research consortium is building a new datacenter. In the datacenter, a set of computers is set up to work together and communicate via a network. The network works only with direct bidirectional links between computers. A pair of computers c_1 and c_2 that are not connected by a direct link can still communicate with each other, as long as there is at least one path of links l_1, l_2, ..., l_k such that links l_i and l_{i+1} share an endpoint, c_1 is an endpoint of l_1, and c_2 is an endpoint of l_k. Any two computers can have at most one direct link between them.

The consortium has asked you to submit a design that illustrates how many computers will be in the network and how they will be connected to each other. Each network design you submit must comply with a specific set of restrictions:

1. There must be between **L** and **U** computers, inclusive, in the network.
2. Each computer must be an endpoint of exactly 4 links, linking it to 4 other distinct computers.
3. Every pair of computers must be able to communicate with each other, as described above.
4. The computers must be able to uniquely identify themselves even if their IDs are randomly changed while the system is off.

To elaborate on the last point: each of the N computers in a network design is initially assigned a unique integer between 1 and N that identifies it. However, it is possible that after some downtime, the system will boot up and the indentifiers will be permuted — that is, each computer will still have a unique integer between 1 and N, but not necessarily the original one. The network must be able to recover the original identifying integers without having access to any information other than the existing direct links.

To evaluate your network designs, the research consortium has set up an automated program. The program will receive one of your network designs, validate conditions 1-3 above, and then send back a copy of the network design with the following changes:

- the unique IDs have been permuted at random (that is, each ID is now equally likely to be on any of the computers),
- every link is listed with the smallest ID first (using the new IDs), and
- the set of links is listed in increasing order of the first endpoint (using the new IDs), breaking ties by smallest second endpoint (i.e., lexicographical order).

You need to be able to determine exactly how the IDs were changed. Formally, the automated program will create a secret random permutation f of the integers 1 through N, and it will assign those numbers to computers in a "blank copy" of the network with all of the previous links removed. Then, for each link between computers i and j in your network design, it will add a link between f(i) and f(j) to the copy. You then must recreate exactly the f that the automated program created. If there exists a different f' that yields the same result and you return f', the consortium will not accept your network design, as in such a case, you cannot ensure that the recovered IDs are the original ones.

For every N between 10 and 100, inclusive, there exists at least one network of N computers that complies with all restrictions above and has the property that applying two different permutations f and f' to it produces two different sets of links.

Input and output

This problem is interactive, which means that the concepts of input and output are different than in standard Code Jam problems. You will interact with a separate process that both provides you with information and evaluates your responses. All information comes into your program via standard input; anything that you need to communicate should be sent via standard output. Remember that many programming languages buffer the output by default, so make sure your output actually goes out (for instance, by flushing the buffer) before blocking to wait for a response. See the FAQ for an explanation

of what it means to flush the buffer. Anything your program sends through standard error is ignored, but it might consume some memory and be counted against your memory limit, so do not overflow it. To help you debug, a local testing tool script (in Python) is provided at the very end of the problem statement. In addition, sample solutions to a previous Code Jam interactive problem (in all of our supported languages) are provided in the analysis for Number Guessing.

Initially, your program should read a single line containing a single integer **T** indicating the number of test cases. Then, you need to process **T** test cases.

For each test case, your program will first read a single line containing two integers **L** and **U** indicating the inclusive range of values for the number of computers in your network design.

Then, you need to create a network design with N computers and print 2N+1 lines representing that design. The first line must contain a single integer N. The remaining 2N lines must contain two integers A and B each, each representing a different link between computers A and B, where A != B. Notice that if you list link A B, you may not list A B nor B A again.

Upon reading your network design, the judge will first check the first three conditions listed in the statement above. If any of those is not met, the judge will send you a single line containing a single `-1`, and then finish all communication and wait for your program to finish. If your program does finish correctly and without violating other limits, it will receive a Wrong Answer verdict.

If all of the conditions are met, the judge will send you 2N+1 lines. The first line will contain a single integer N (the same N you sent). Then, the next 2N lines will contain two integers each, describing the links of the copy of the network design, in the same format as you used. The copy is generated as described above, with the permutation f chosen uniformly at random from all possible permutations, independently of your network design.

To finish a test case, you need to send the judge a single line with N integers X_1, X_2, ..., X_N, representing that the computer to which you assigned number i was assigned number X_i in the judge's copy, for all i.

If the list is not the list the judge generated, you will receive a Wrong Answer verdict. If it was in the last test case, the judge will send no additional communication. Otherwise, the judge will send a single line containing a single -1, and then no additional communication. In both cases, the judge will wait for your program to end, and assign the Wrong Answer verdict only if it ended normally and without violating any resource limits.

You should not send additional information to the judge after solving all test cases. In other words, if your program keeps printing to standard output after printing the list of Xs for the last test case, you will receive a Wrong Answer judgment.

Notice that you are allowed to submit the same network design for different test cases, as long as that design complies with all restrictions for both cases. Additionally, the seed from random generation in the judge is fixed, so sending the same set of original network designs in the same order will get back the same set of copies.

Limits

$1 \leq \mathbf{T} \leq 30$.
Time limit: 10 seconds per test set.
Memory limit: 1GB.

Test set 1 (Visible)

L = 10.
U = 50.

Test set 2 (Hidden)

$10 \leq \mathbf{L} \leq 50$.
L = **U**.

Sample Interaction

Note that this sample interaction uses a smaller value of **L** than the real data, for ease of illustration. Also note that there is no network of exactly 6 computers with the property that applying two different permutations f and f' to it produces two different sets of links, so it would be a bad idea to design a network of exactly 6 computers, even if the problem's limits allowed it!

```
t = readline_int()            // reads 2 into t
limits = readline_int_list()  // reads 6 50 into limits
printline 6 to stdout         // using 6 computers. Contestant designs an
                              //   octahedral network
printline 2 4 to stdout
printline 1 2 to stdout       // you do not need to list edges in any
                              //   particular order
printline 3 1 to stdout       // you do not need to give the endpoints of a
                              //   link in order
printline 1 4 to stdout
printline 1 5 to stdout
printline 2 3 to stdout
printline 2 6 to stdout
printline 3 5 to stdout
printline 3 6 to stdout
printline 4 5 to stdout
printline 4 6 to stdout
printline 5 6 to stdout
flush stdout                  // judge verifies that the network meets
                              //   conditions 1-3 above, and secretly picks
                              //   2 6 3 1 5 4 as the new permutation
n = readline_int()            // reads 6 into n
repeat 12 times:
  add readline_int_list() to edges  // reads 1 2, 1 4, 1 5, 1 6, 2 3, 2 5,
                                    //   2 6, 3 4, 3 5, 3 6, 4 5, 4 6, in
                                    //   that order
printline 2 5 6 1 3 4         // note that this is consistent with what the
                              //   judge sent, but is not the permutation the
                              //   judge used
flush stdout                  // judge decides that this is wrong
limits = readline_int_list()  // expects to read the next case but gets -1,
                              //   indicating a wrong answer
exit                          // exits to avoid an ambiguous TLE error
```

Testing Tool

You can use this testing tool to test locally or on our platform. To test locally, you will need to run the tool in parallel with your code; you can use our interactive runner for that. For more information, read the instructions in comments in that file, and also check out the Interactive Problems section of the FAQ.

Instructions for the testing tool are included in comments within the tool. We encourage you to add your own test cases. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently. If your code passes the testing tool but fails the real judge, please check the Coding section of the FAQ to make sure that you are using the same compiler as us.

Download testing tool