Problem proposal: Toy Design

Petr Mitrichev (petr.mitrichev@gmail.com)

February 19, 2022

1 Problem Statement

You are working for a company that designs toys. A new toy that is being created works like this: there are n pins, numbered from 1 to n, sticking out of a box, and some pairs of pins are connected with wires inside the box (in other words, the pins and wires form an undirected graph). The wires are not visible, and the only way to find out something about them is to use a *tester* on the pins: we can pick two pins i and j ($i \neq j$), and the tester will tell if those two pins are connected directly or indirectly inside the box (in other words, whether there is a path between those pins in the graph).

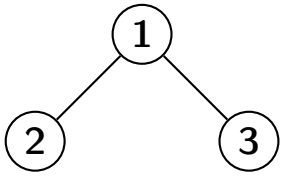
We will call the set of connections inside the box the *design* of the toy.

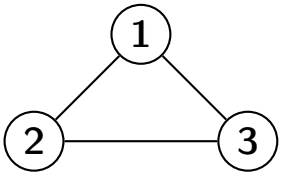
You are using specialized software to create those designs. This software works like this: it starts with some design of the toy which we denote "design 0". It does not show you the connections inside the box for this design. Instead, you can repeatedly do the following operation:

- You pick a design number a and two pin numbers i and j ($i \neq j$).
- The software tells you what would happen if we use the tester on those two pins. In other words, it tells you if pins i and j are directly or indirectly connected in design a.
- Also, if the pins were not directly or indirectly connected in design a, then it creates a new design which has all connections from design a plus one additional connection between i and j, and gives this design the next available number (so the first design created in this way will have number 1, then number 2, and so on). Note that this does not change design a, just creates a new design that has the additional connection.

Your goal is to learn as much as possible about design 0 by using this operation.

Note that is not always possible to determine the exact set of connections for design 0, because there is no way to distinguish direct and indirect connections. For example, consider the following two designs with $n = 3$:





The tester would report that any two pins are connected for each of the designs, so we won't be able to distinguish them using the software described above.

Therefore your goal is to determine any design that is *equivalent* to design 0. Two designs are *equivalent* when for any two pins i and j the tester reports the same result for both designs (either they are directly or indirectly connected in both designs, or they are not directly or indirectly connected in both designs).

This is an interactive problem.

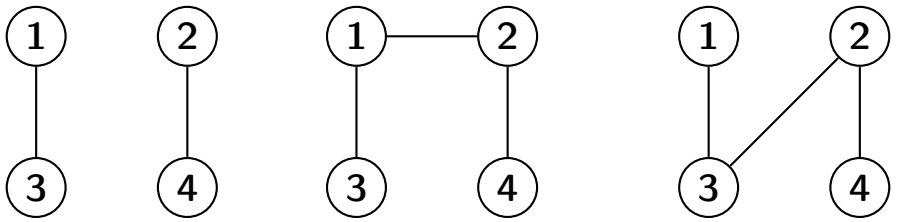
First, your program should read a single integer n ($n \geq 2$). Then it should do zero or more operations. In one operation, it should first print `?  a i j` ($0 \leq a$, $1 \leq i, j \leq n$, $i \neq j$, a must not exceed that maximum number of a design created so far). Then it should flush the standard output stream. Then it should read a single integer b. In case pins i and j were already directly or indirectly connected in design a, then you will get the integer a back: $b = a$. Otherwise b will be the number assigned to the new design which has all connections of design a plus the connection between i and j. In this case b will always be equal to the maximum number of a design created so far plus one.

When your program is done doing the operations, it should describe a design that is equivalent to design 0 and then finish its execution gracefully. To describe a design, it should first print `!  m` ($0 \leq m \leq \frac{n(n-1)}{2}$, where m denotes the number of direct connections in your design. Then, it should print m lines, each containing two pin numbers, describing the connections. There must be at most one connection between every pair of pins, and no connections connecting a pin to itself.

Example interaction:

```
4
?  0 1 2
1
?  1 2 3
1
?  0 2 3
2
?  1 3 4
1
!  2
1 3
2 4
```

Here are the designs involved in this interaction:



Design 0 Design 1 Design 2

Initially, we have just design 0. Then we ask if 1 and 2 are connected in design 0, they are not, so a new design 1 is created where they are connected. Then we ask if 2 and 3 are connected in design 1, and they are (indirectly through 1), so no new design is created. Then we ask if 2 and 3 are connected in design 0, and they are not, so a new design 2 is created where they are connected. Then we ask if 3 and 4 are connected in design 1, and they are (indirectly through 1 and 2). Finally, we are ready to describe a design that is equivalent to design 0 — in fact, in this case our output matches design 0 exactly.

2 Subtasks and anticipated difficulty

Here k denotes the number of operations that the solution is allowed to do.

- Subtask 1: $n \leq 200$, $k = 20000$. Here we can just query all pairs, very straightforward.
- Subtask 2: $n \leq 8$, $k = 20$. Here we need to improve on the trivial algorithm (which would have $k = 28$), either by some casework, or by bruteforce search of all possible decisions, or by applying the solution from the next subtask.
- Subtask 3: $n \leq 200$, $k = 2000$. The solutions with the right asymptotic number of operations, but with a larger constant. We can tweak the constraint here after implementing several such solutions.
- Subtask 4: $n \leq 200$, $k = 1350$. The actual bound on the number of operations is $\sum_{i=1}^{n} \lceil log_2 i \rceil$. See the intended solution below.

We can also try to squeeze a lower constant than $\sum_{i=1}^{n} \lceil log_2 i \rceil$ to create an even harder subtask, but I don't think it would add much to the problem.

We also need to pay attention to making sure that solutions with a suboptimal constant in $O(n \log n)$ pass subtask 3 but not subtask 4. In case this turns out to be hard or impossible to achieve with the test data, we can merge the two subtasks into one (with the constraints of subtask 3).

3 Intended solution

Let us process pins one by one from 1 to n, and after processing pin k we will know the connected components of pins $1, 2, \ldots, k$, as well as have designs where the first i connected components are merged into one, for each i.

Suppose we have u connected components from pins $1, 2, \ldots, k$ and are procesing pin $k+1$. Let's denote as $f(i)$ the following boolean function: is pin k connected to pin 1 if we merge the first i connected components of the first k pins together. Since we already have designs for all prefix merges, we can evaluate $f(i)$ in one operation. It's also clear that $f(i)$ is monotonic, so we can use binary search to find the smallest value of i such that $f(i)$ is true, or to determine that there is no such value. There are $u + 1$ possible outcomes of this binary search, so we need $\lceil log_2(u + 1) \rceil$ operations for it.

It's not hard to see that this value of i will be exactly the number of the connected component that the vertex $k+1$ is connected to in design 0 (and if there is no such i, then it starts a new connected component), and in case there is no such i, we will also get a new design where this new component is merged to pin 1 as a side effect, so we need no additional operations for that.

And since $u+1 \leq k+1$, the total number of operations is at most $\sum_{i=1}^{n} \lceil log_2 i \rceil$.

Note that the "specialized software" described in the problem statement implements a well-known data structure, *persistent disjoint-set union*. However, since the contestants do not need to implement it, I think the knowledge of this data structure does not really help in solving the problem, therefore this problem is good for EGOI.

4 Suggestions for grading

I think we should assign a meaningful amount of points to subtask 2, to encourage participants to experiment for small values of n if they can't find a general solution. So maybe something like:

- Subtask 1 — 10 points
- Subtask 2 — 20 points
- Subtask 3 — 35 points
- Subtask 4 — 35 points

5 Thoughts about test data

I think just random graphs with varying amount of connected components should work pretty well here, but we can also try to have graphs with many isolated vertices at the beginning or at the end plus a few large connected components.

Another set of cases that should catch many suboptimal solutions are graphs where all components have size 1 or 2.

In general we need to implement a few solutions with a suboptimal constant in $O(n \log n)$ and see how to separate them.

We can consider having multiple testcases in one program run to have better test coverage, it depends on the details of how the interactive problems are supported in the judging system.

6 Contact details

Petr Mitrichev

e-mail: petr.mitrichev@gmail.com

Affiliation: Software Engineer at Google

Country: Switzerland