Sopsug

EGOI 2023

Problem author: Jakub Tarnawski.

1 The Problem

You are given a graph with N nodes, M good directed edges and K bad (forbidden) directed edges. Build a directed tree with any root, such that all edges are directed towards the root, all good edges are used, and none of the bad (forbidden) edges are used.

We denote by (u, v) an edge that is directed from u to v.

2 Subtask 1: $M = 0, K = 1$

This is an ad-hoc subtask; its solution is not particularly instructive for the full solution.

There aren't any edges that you must use, and there is exactly one bad edge that you can't use. Consider the following two trees, both of which are directed chains:

- A tree such that all edges are $(i, i + 1)$. (Think about "going right" if all nodes are placed on a line.)
- A tree such that all edges are $(i + 1, i)$. ("Going left".)

At least one of the trees does not contain the bad edge. You can just figure out which one, and output it.

3 Subtask 2: $M = 0, K = 2$

This is an ad-hoc subtask; its solution is not particularly instructive for the full solution.

Similar to Subtask 1, you may consider finding a directed chain such that the edges in the directed chain do not contain the bad edges. An easy way to achieve this without much thinking is to generate some random chains, then

check whether the generated chain works. Since the number of bad edges is small, after a few iterations there will be a chain that satisfies the conditions.

There is a special case: when $N = 2$, there is no solution.

Another solution is to find a node that has no incoming bad edge; since $K = 2$, if $N \geq 3$ there must be such a node. Then connect all other nodes directly to that node.

4 Subtask 3: $K = 0$

There are no bad edges, so one simply needs to construct a directed tree that uses all the good edges.

If the good edges form a directed forest, then it is possible. Otherwise, it is impossible. The exact conditions for the edges to form a directed forest are:

- Each node should have out-degree at most 1.
- There should be no cycles.

One can check acyclicity using a graph traversal method such as DFS. Alternatively, one can use a Union-Find data structure; whenever processing a new edge (u, v), first check if u and v are not already in the same component (in which case this is not a directed forest).

If the answer is "possible", simply pick one of the trees in the forest and connect the remaining trees' roots to any node in that tree.

5 Subtask 4: $N \leq 100$

We begin from the solution to Subtask 3. First, note that the good edges must still form a directed forest for there to be a solution. Next, we look at the resulting forest (where some trees are perhaps just single nodes). It is not hard to see that any final solution must have the following shape: we must choose one of the trees in the forest as the "root", and then connect each other tree in the forest to the "root" tree (perhaps indirectly). Note that any new edge will need to go from *the root* of a tree to *any node* of another tree.

We will try every possible tree as the "root" tree. So let us fix a "root" tree. We perform a DFS starting from the root of the "root" tree. Note that edges have to be reversed in order to be able to traverse the tree. When we visit a node, we try to connect any of the roots of trees that aren't connected yet. This means checking if the appropriate edge is not forbidden. If we succeed, great; we should then also proceed with the DFS from the newly-connected tree root. If we do not succeed, then we have seen a forbidden edge.

What is the time complexity of this, for a fixed "root" tree? Note that we traverse tree edges at most $N - 1$ times (each traversal corresponds either to an

existing forest edge, or to a new edge with which we "succeed" in connecting a new tree root). Moreover, we make at most K attempts to connect a new tree root that do not "succeed" because the considered edge was forbidden. Each operation should be dominated by the cost of checking if an edge is forbidden, which should be at most $O(\log N)$.[1] Therefore we have $O((N+K)\log N)$ total, per "root" tree.

Finally, checking every possible "root" tree contributes another N factor, which is good enough for this subtask.

As we actually have $N \leq 100$ here, one can even afford to explicitly construct the entire "complement" graph consisting of non-forbidden edges (of which there are at most $N^2 - K$) and run DFS on it appropriately. The time complexity of such an approach would be $O(N^3)$.

6 Subtask 5: Exists solution with 0 as root

As we only need to check one "root", we do not incur the extra N runtime factor, so the solution from Subtask 4 is fast enough even for $N = 300\,000$. (However, explicitly constructing the complement graph would no longer be possible, and one has to be somewhat careful not to run into some quadratic-time behavior.)

7 Full task

We need to speed up the solution from Subtask 4. What is a good "root", actually? It might be easiest to first think of the case where there are no good edges ($M = 0$). Then, a good "root" is a node from which every node is reachable in the complement graph (i.e., the complete graph from which all forbidden edges have been removed, and then all remaining edges were reversed). Suppose we run DFS from node $v_0 := 0$; once it's done, check if every node was visited. If yes, then 0 was a good root. Otherwise, run another DFS from some yet-unvisited node v_1, without resetting the "visited" status of any nodes. Continue doing this: in each iteration, if not all nodes have been visited, run another DFS from some yet-unvisited node, and so on. Eventually we will have visited all nodes. Let us pay special attention to the last node v_k from which we ran DFS.

Is v_k a good root? Well, perhaps not – it might, for example, be an isolated node. However, surely every node that was visited in previous DFS iterations (before we started from v_k) was *not* a good root, as v_k was not visited in these iterations. On the other hand, all the other nodes – those first visited in the last DFS iteration – are reachable from v_k. Therefore, either v_k is a good root, or there is no solution. To check whether v_k is a good root, we reset the "visited" status of every node, and run one last DFS from v_k (building the directed tree if possible).

[1] This can be brought down to $O(1)$, e.g. using a hash-set, such as `unordered_set` in C++.

This solution idea can be seen as being inspired by Kosaraju's algorithm for finding strongly connected components of a directed graph (the one where we run a DFS, and then another DFS on the graph with reversed edges).

Now, when $M > 0$, for the proof of correctness one can think of contracting every tree in the forest (compressing it into one node) while handling the edges appropriately. For the implementation, we just run the solution from Subtask 5, but then proceed with further DFS runs if not all nodes were visited from 0, as above.

The time complexity analysis goes through just as for Subtasks 4–5.

Subtask 6 ($M = 0$) is used as a safety net for the contestants who had the idea for the full solution, but implemented something wrongly for the good edges, e.g. checking acyclicity.