Solution 3:

nitial thoughts

- Convert every undirected road to a one-way road while keeping “reachable from any junction to any other” ⇒ the directed graph must be **strongly connected**.
- Plan: run DFS, orient edges as I see them, and print the directions.

What went wrong

- I oriented edges naively during an iterative DFS. Some “return paths” were missing, so node 1 (for example) became unreachable from parts of the graph.
- I didn’t check for **bridges** first. If the undirected graph has a bridge, no orientation can be strongly connected, printing anything but `0` is wrong.
- I used hash maps/sets (`unordered_map` for adjacency, `unordered_set<pair<int,int>>` for edges). This added overhead and made direction bookkeeping error-prone.
- I briefly used variable-length arrays and `memset` on `vector`, which caused portability/compile issues.

Why it failed

- In an undirected DFS, each edge appears twice. Without a strict “process each undirected edge exactly once” rule, directions can be duplicated or missed.
- Treating some non-tree edges in the wrong direction breaks cycles needed for strong connectivity.
- Skipping the bridge check lets impossible cases slip through.

What I fixed

1. **Bridge test first (Tarjan)**

   Run lowlink on the undirected graph. If any edge is a bridge, output `0` immediately. This enforces the necessary condition: the graph must be **2-edge-connected**.

2. **Edge IDs + single processing**

   Assign every input edge an **ID** and keep a `used[id]` flag. This guarantees each undirected edge is oriented **exactly once**, regardless of which endpoint discovers it.

3. **Consistent orientation rule**

   Do a DFS from any node. For every adjacency `(u → v)` you traverse:

   - If `v` is **unvisited** (tree edge): orient `u → v` and recurse.
   - If `v` is **already visited** (back/cross): still orient `u → v`.

     This creates the necessary cycles (there are always edges going "forward" to visited ancestors), and with no bridges, the result is strongly connected.

4. **Dense structures & safe containers**

   Swapped `unordered_map<int,vector<...>>` for `vector<vector<...>>` (nodes are 1..n), removed pair hashing, avoided VLAs and `memset` on `vector`.

Result

- Correctness: prints `0` when a bridge exists; otherwise yields a valid strong orientation.
- Simpler reasoning: each edge handled once via its ID; directions are uniform.
- Performance: linear time **O(n+m)** with small constants, good cache behavior.

Takeaway

Strong connectivity after orientation isn't about clever cycles post-hoc; it's about (1) **no bridges**, and (2) a **deterministic once-per-edge orientation** that naturally forms cycles during DFS. Everything else (maps, pair hashes, iterative quirks) just added noise.