Failover

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	2048 MB

Your company operates a datacenter with N computers ("nodes") running S services (web servers, databases, caches, microservices, etc). The services form a directed acyclic graph with L layers. Layer 0 is the ingress layer that clients hit.

You are building an automated drain controller. It periodically samples the external traffic rate at each layer-0 service, and must figure out which nodes are overloaded and pull the plug on them. It does this in two steps: first it decides which nodes to drain based on load, then it updates the service topology to reflect the drains.

Services

Each service belongs to one layer (0 through $L-1$) and has a fixed **origin node**. An edge $u \to v$ with weight w means every request reaching service u produces w requests to service v. Edges only go from layer k to layer $k+1$.

Nodes and failover

Each node i has a capacity cap_i and a failover target fail_i. The failover target is either another node or 0 ("the void" — traffic is dropped).

When a node is drained, it is drained forever. To find where a service runs, start at its origin node and follow the failover chain until you reach a live node — that node is the service's **host**. If the chain reaches the void **or a drained node** without hitting a live node, the service is **stopped**: it has rate 0 and does not forward anything.

Traffic flows only through alive services. An alive ingress service receives its external rate. An alive non-ingress service receives the sum of $w \cdot \text{rate}(u)$ over all alive predecessors u with edge weight w. The **load** on a live node is the total rate of all alive services hosted on it.

What happens on each operation

When the controller processes an operation that changes traffic or drains a node, it runs two steps:

1. **Drain overloaded nodes.** Compute the load on each live node once from the service graph as it stands at the start of this step. If a live node's load exceeds its capacity, drain it. When a node is drained, its load moves to whatever live node its failover chain reaches (or is lost to the void). This may overload another node, so repeat until no live node is overloaded. Draining is permanent. The service graph is *not* re-evaluated during this step — loads are only moved via failover.
2. **Update the topology.** Some services may now be stopped because the drains in step 1 made their entire failover chain collapse to the void. Recompute the load on every live node from the service graph, now excluding stopped services. This can only lower loads, so no further drains happen.

The order matters: step 1 decides which nodes to drain using the load picture from *before* any services stop. Step 2 cleans up afterward.

Operations (Q total)

- `1` s x (TRAFFIC) — set ingress service s's external rate to x ($1 \le s \le S$, $1 \le x \le 10^5$; s must be a layer-0 service). Run both steps. Print the load hash.
- `2` i (DRAIN) — force-drain node i ($1 \le i \le N$). Run both steps. Print the load hash. Node i is not necessarily still alive.
- `3` i (SIMULATE) — hypothetically drain node i ($1 \le i \le N$) and run *only step 1* on a snapshot. Node i is not necessarily still alive. Print how many nodes drain in total (including i). Then throw away all changes.

Load hash

After each TRAFFIC or DRAIN, let load(n) be node n's load if it is live, or 0 if it is drained. Let m be the total number of drained nodes. Print

$$H \;=\; \Big(m \;+\; \sum_{n=1}^{N} x^n \cdot \mathrm{load}(n) \Big) \bmod p\,, \qquad p = 10^9 + 7, \quad x = 10\,007\,.$$

Input

The first line contains N ($1 \le N \le 20$).

The second line contains N integers $\mathrm{cap}_1, \ldots, \mathrm{cap}_N$ ($1 \le \mathrm{cap}_i \le 10^9$).

The third line contains N integers $\mathrm{fail}_1, \ldots, \mathrm{fail}_N$ ($0 \le \mathrm{fail}_i \le N$, $\mathrm{fail}_i \ne i$). The failover graph may contain cycles.

The next line contains L ($1 \le L \le 6$).

The next line contains S ($1 \le S \le 2 \cdot 10^4$).

The next S lines each contain two integers: the layer ($0 \le \text{layer} < L$) and the origin node ($1 \le \text{node} \le N$) of service v.

The next line contains E ($1 \le E \le 2 \cdot 10^5$), followed by E lines each with three integers u, v, w ($1 \le u, v \le S$, $\mathrm{layer}(v) = \mathrm{layer}(u) + 1$, $1 \le w \le 4$).

The next line contains Q ($1 \le Q \le 2 \cdot 10^5$), followed by Q operations as described above.

It is guaranteed that every load value that arises during the computation — including mid-step transient values before drains settle — is at most 10^{15}, so all arithmetic fits in signed 64-bit integers.

Output

For each TRAFFIC or DRAIN, print one line with the hash H. For each SIMULATE, print one line with the number of nodes drained.

Example

standard input	standard output
4 100 10 100 1000 2 0 4 0 3 3 0 1 1 2 2 3 2 1 2 1 2 3 1 5 1 1 5 1 1 12 3 1 2 3 1 1 20	8066918 120085 1 120086 200142

Explanation: Three services form a chain $A \to B \to C$ (all multipliers 1) on nodes 1, 2, 3. Node 4 is a spare. Failover: $1 \to 2$, $2 \to$ void, $3 \to 4$, $4 \to$ void.

- `1 1 5`: every service gets rate 5. Loads are $(5, 5, 5, 0)$. No node is overloaded.
- `1 1 12`: every service gets rate 12. Loads are $(12, 12, 12, 0)$. Node 2 has load $12 >$ cap $= 10$, so it is drained. Its failover is the void, so its load is dropped. After draining, service B (origin node 2) has no live host — it stops, and C gets no traffic. Only node 1 keeps load 12.
- `3 1`: hypothetically drain node 1. Its failover chain goes to node 2 (already drained) then to void. No other node is overloaded. One node drained total.
- `2 3`: force-drain node 3. Its load was already 0, so the load (0) moves to node 4. Nothing changes.
- `1 1 20`: node 1 is the only one carrying load. Load is 20.