MAC Learning

HackerRank

Problem Statement

Introduction

Early Ethernet networks were connected by hubs, which output each packet received on a given port to every other port on the hub. This is effective but wastes network bandwidth because every packet propagates (directly or indirectly) to every host attached to the network. Later, Ethernet switches were introduced. A switch uses the Ethernet "MAC learning" algorithm to propagate packets only to the switch port that is the packet's actual destination. This problem consists of building a simple implementation of the MAC learning algorithm.

Let's start with some important definitions. An Ethernet address, or "MAC address", is a 48-bit (6-byte) number that is canonically represented by six pairs of hex digits delimited by colons. Every NIC should have a globally unique MAC. MAC addresses have a little bit of structure, and in particular the least-significant bit of the first byte of a MAC is 0 if the address represents a single host or 1 if the address is a special "multicast" address that represents multiple hosts. Thus, 00:26:b9:8c:b0:f9 (the Ethernet address for my workstation's NIC) is a unicast address, because 00 has 0 as its least-significant bit, whereas ff:ff:ff:ff:ff:ff is a multicast address (specifically the special "broadcast" address).

An Ethernet packet, or "frame", begins with a standardized 14-byte header. The first 12 bytes of the header are a pair of Ethernet addresses: first, the destination Ethernet address, then the source Ethernet address. (The source address is always a unicast address.) The last 2 bytes are the Ethernet type, which describes the kind of data that follows the header. For simple MAC learning, only the source and destination addresses are significant.

Algorithm

MAC learning uses a dictionary M, that maps from an Ethernet address to a switch port. The dictionary is initially empty, and its contents persist from one invocation of the algorithm to the next. Suppose that the switch receives a packet on port Pin that has source address S and destination address D. The switch executes the following algorithm:

- If S is a multicast address, terminate the algorithm with output "drop," indicating that the packet should be dropped.
- Add the mapping S -> Pin to M. If there is already a mapping from S to a different port, update the mapping; if S -> Pin already exists in M, no change is necessary.
- If D is a multicast address, terminate the algorithm with output "flood," indicating that the packet should be forwarded to all of the switch's ports except for port Pin.
- Search M for a mapping for D. If there is no such mapping, terminate the algorithm with output "flood" (the switch doesn't know which port to use, so it outputs it on all the ports). Otherwise, let Pout be the value found in M.
- If Pin = Pout, terminate the algorithm with output "drop," because a switch should not send a packet back out the port it came in on.
- Terminate the algorithm with output Pout.

Input and Output Format

Input begins with a single line that indicates N, the number of packets to be processed by the algorithm. Each of the N subsequent lines represents a packet to be processed and consists of an input port number Pin (as a decimal number), a space, the destination Ethernet address D, a space, and the source Ethernet address S, each expressed as 6 pairs of hex digits delimited by colons.

The output consists of N lines, each of which is one of the strings "drop" or "flood" (without the quotes) or a decimal number (representing Pout).

Constraints

Each input port number Pin is in the range 0 to 255, inclusive.

Dictionary M need not hold more than 16 mappings at a time.

Sample Test case

Consider the following input:

```
7
1 10:a3:fe:8b:a7:2c 08:6e:90:55:3a:97
2 08:6e:90:55:3a:97 10:a3:fe:8b:a7:2c
1 10:a3:fe:8b:a7:2c 08:6e:90:55:3a:97
1 12:f6:91:9c:6f:0c 01:ac:f0:27:c0:2e
2 10:a3:fe:8b:a7:2c 08:6e:90:55:3a:97
3 01:ac:f0:27:c0:2e 10:a3:fe:8b:a7:2c
2 10:a3:fe:8b:a7:2c 08:6e:90:55:3a:97
```

The expected output is:

```
flood
1
2
drop
drop
flood
3
```

These lines of output are generated by steps 4, 6, 6, 1, 5, 3, and 6, respectively.