You are encouraged to work in groups of two or three. If you cannot find partners, you may work alone. Please submit **one** copy of the assignment using `subversion`; if you are working with partners, all names should appear in the Python code (as comments). Note that you will get a grade of zero if your program does not run.

The first ARPANET link[1] was established between the University of California, Los Angeles, and the Stanford Research Institute, at 10:30pm on October 29, 1969. Stanford University continues being on the cutting edge of "Computer and Network Security" research, and one of the important tools to come out from that environment is OpenFlow[2]. Here is a summary of what it is: *OpenFlow is an open interface for remotely controlling the forwarding tables in network switches, routers, and access points. Upon this low-level primitive, researchers can build networks with new high-level properties. For example, OpenFlow enables more secure default-off networks, wireless networks with smooth handoffs, scalable data center networks, host mobility, more energy-efficient networks and new wide-area networks — to name a few.*

In this assignment you are going to implement a mini-version simulation in Python of one aspect of OpenFlow.

A *hub* is a network switch with a given number of ports. When a hub receives a packet on port p, it forwards it to all ports q except for the incoming port p. That is, if a hub has ports 1 through 128, and it receives a packet on port 74, it forwards this packet to ever port $q \in [128] - \{74\}$. A *Learning Switch* is a bit smarter; when a packet is received, the switch "learns" the MAC-port association. For example, if a packet with source MAC-address `FF:B3:5A:32:DF:EE` comes in port 2, we know that subsequent packets destined to that address should be forwarded to port 2 rather than all the ports. Your simulator will take as input a text file of the form:

```
128
32
51 2C:37:BB:25:74:F8 B8:97:D7:45:34:FF
32 AA:F1:35:C5:94:E3 88:87:47:DE:D4:32
12 AB:E2:46:2C:78:11 AA:F1:35:C5:94:E3
...
```

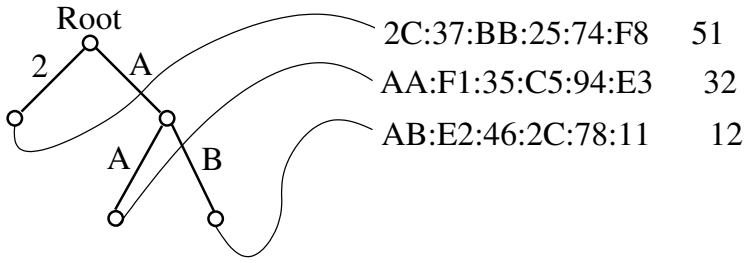
where in each line the triple p, s, d represents the port of the switch, the source MAC-address and the destination MAC-address. However, the first two lines are special: the first line specifies the number of ports of the switch, and the second line specifies the size of the cache of the switch.

As your simulator processes the input file line by line, it is going to be maintaining a data structure that is called a *hexadecimal trie*. This is a tree meant to help identifying the ports

[1] The Advanced Research Projects Agency Network (ARPANET), was the world's first operational packet switching network and the core network of a set that came to compose the global Internet.

[2] `http://www.openflow.org/`

associated with source addresses that have been processed already and are in the cache. The trie keeps track of the shortest prefix necessary to identify uniquely each address in the cache. The leaves of the hexadecimal trie will be pointing to entries in the cache which will contain pairs (addresses, port), in a least-recently-used ordering. After processing the first 3 requests as in the example file, the data structure would look like follows:



Note that `2C:37:BB:25:74:F8` is the least-recently-used address at this point. While processing the input file, if a source or destination in a given line is already in the cache, it should be put in the most-recently-used position (which in this diagram would be the bottom). If the list becomes longer than the size of the cache (32 in our example), then the least-recently-used entry (the top in our diagram) would be removed, and the new entry put in the most-recently-used position; the trie has to be updated accordingly. Note that the position of an address in the list changes whether it appears as source or destination in the input, but only the sources are used to compute the hexadecimal trie. The purpose of the hexadecimal trie is to locate quickly an address and its corresponding port.

This describes the data structure maintained by the switch. In your simulation, after processing each line (except the first two), you should output the path in the trie to the destination (together with the port), or say "all ports" if there is no record of the destination address in the trie. In our running example:

```
B8:97:D7:45:34:FF all ports except 51
88:87:47:DE:D4:32 all ports except 32
AA:F1:35:C5:94:E3 Root --> A --> AA port 32
...
```

Make a design decision regarding what happens if a particular MAC-address is discovered to be associated with two ports; will such ambiguous port assignments be always discovered? Also make a design decision regarding how to update a trie efficiently after deleting the least-recently-used entry to make room for a new address in the cache.

Please include a text write up of your program. That is, write a short report (a few paragraphs, included as a comment at the beginning of your code) describing how you solved the problem, how was your program tested, any problems you encountered, a sample run of the program (and an output dump of the sample run), and a justification of your design decisions. The intention is to have a basic "manual" for your program.