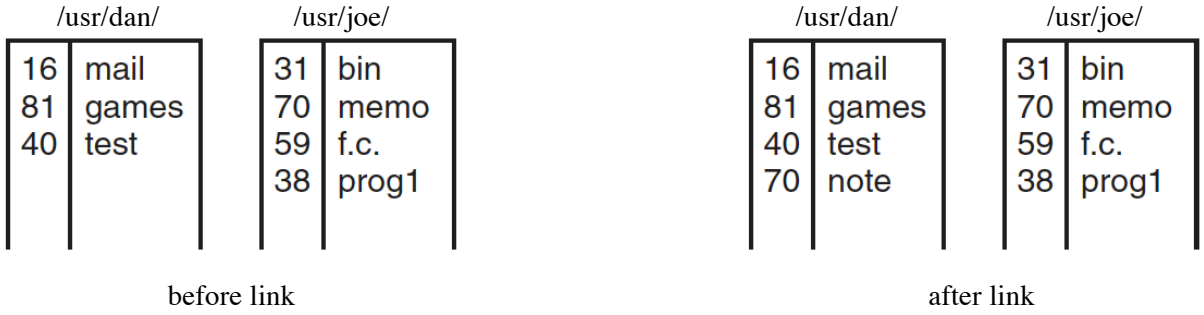
Link

All operating systems that run on personal computers have system calls for directory management. One such call is *link*. Its purpose is to allow the same file to appear under two or more names, often in different directories. A typical use is to allow several members of the same programming team to share a common file, with each of them having the file appear in his own directory, possibly under different names. Sharing a file is not the same as giving every team member a private copy; having a shared file means that changes that any member of the team makes are instantly visible to the other members—there is only one file taking up space on disk. To see how *link* works, consider the situation in the figure below. Here are two users, joe and dan, each having his own directory with some files. If joe now executes a program containing the system call

link("/usr/joe/memo", "/usr/dan/note");

the file *memo* in joe's directory is now entered into dan's directory under the name *note*. Thereafter, /usr/joe/memo and /usr/dan/note refer to the same file.




before link  after link

Every file in UNIX has a unique number, its i-number, that identifies it. This i-number is an index into a table of i-nodes, one per file, telling who owns the file, where its disk blocks are, and so on. A directory is simply a file containing a set of (inode-number, ASCII name) pairs. In the above example, if either file is later removed, the other one remains. If both are removed, UNIX sees that no entries to the file exist (a field in the i-node keeps track of the number of directory entries pointing to the file), so the file is removed from the disk. One can create, remove, and link files in an UNIX based operating system by accessing a terminal and running the commands touch, rm, and link respectively.

Your task is to implement the i-node system. Should an i-node or more come to a state where they point to no files, whenever the touch command is executed again, that file shall be mapped to the smallest i-node having no files at the time. Should there be no available i-nodes to attach a file to, you will create a new i-node in the directory. To simplify your task, you only have to implement the i-nodes functionality in a single folder, not across different folders.

**Input**

The input begins with a single positive integer on a line by itself indicating the number of test cases to follow. For each test case, there is number N indicating the number of commands in a terminal to follow. Following this are N lines, each line in one of the following formats:

| | |
|---|---|
| ***touch filename*** | ***(to create a file and name it filename)*** |
| ***link filename filename*** | ***(to link two files in a directory)*** |
| ***rm filename*** | ***(to remove a file)*** |

**Output**
For each test case, output all i-nodes created for the specified input and a corresponding list of all different filenames still mapped to that i-node in lexicographical order in the format below or nothing if no files are mapped to that node. There must be one blank line between each test case output.

***Node 0: filename***
***Node 1: filename filename filename***
***Node 2:***
***Node 3: filename***

**Sample Input**

```
5

3
touch a.txt
touch b.txt
rm a.txt

4
touch LinkSolution.java
touch YouWish.txt
rm LinkSolution.java
touch funny.txt

6
touch z.txt
link z.txt a.txt
link a.txt y.txt
link y.txt b.txt
link b.txt u.txt
rm y.txt

6
touch b.txt
link b.txt a.txt
touch c.txt
touch d.txt
link d.txt e.txt
rm c.txt

4
touch a.txt
link a.txt b.txt
link b.txt a.txt
rm a.txt
```

**Sample Output**

```
Node 0:
Node 1: b.txt

Node 0: funny.txt
Node 1: YouWish.txt

Node 0: a.txt b.txt u.txt z.txt

Node 0: a.txt b.txt
Node 1:
Node 2: d.txt e.txt

Node 0: b.txt
```