Problem C

It's not a Bug, it's a Feature!

Source: `bugs.(c|cc|pas|java)`
Input: `bugs.in`

It is a curious fact that consumers buying a new software product generally do *not* expect the software to be bug-free. Can you imagine buying a car whose steering wheel only turns to the right? Or a CD-player that plays only CDs with country music on them? Probably not. But for software systems it seems to be acceptable if they do not perform as they should do. In fact, many software companies have adopted the habit of sending out patches to fix bugs every few weeks after a new product is released (and even charging money for the patches).

Tinyware Inc. is one of those companies. After releasing a new word processing software this summer, they have been producing patches ever since. Only this weekend they have realized a big problem with the patches they released. While all patches fix some bugs, they often rely on other bugs to be present to be installed. This happens because to fix one bug, the patches exploit the special behavior of the program due to another bug.

More formally, the situation looks like this. Tinyware has found a total of n bugs $B = \{b_1, b_2, \ldots, b_n\}$ in their software. And they have released m patches p_1, p_2, ..., p_m. To apply patch p_i to the software, the bugs $B_i^+ \subseteq B$ have to be present in the software, and the bugs $B_i^- \subseteq B$ must be absent (of course $B_i^+ \cap B_i^- = \emptyset$ holds). The patch then fixes the bugs $F_i^- \subseteq B$ (if they have been present) and introduces the new bugs $F_i^+ \subseteq B$ (where, again, $F_i^- \cap F_i^+ = \emptyset$).

Tinyware's problem is a simple one. Given the original version of their software, which contains all the bugs in B, it is possible to apply a sequence of patches to the software which results in a bug-free version of the software? And if so, assuming that every patch takes a certain time to apply, how long does the fastest sequence take?

Input

The input contains several product descriptions. Each description starts with a line containing two integers n and m, the number of bugs and patches, respectively. These values satisfy $1 \leq n \leq 20$ and $1 \leq m \leq 100$. This is followed by m lines describing the m patches in order. Each line contains an integer, the time in seconds it takes to apply the patch, and two strings of n characters each.

The first of these strings describes the bugs that have to be present or absent before the patch can be applied. The i-th position of that string is a "+" if bug b_i has to be present, a "−" if bug b_i has to be absent, and a "0" if it doesn't matter whether the bug is present or not.

The second string describes which bugs are fixed and introduced by the patch. The i-th position of that string is a "+" if bug b_i is introduced by the patch, a "−" if bug b_i is removed by the patch (if it was present), and a "0" if bug b_i is not affected by the patch (if it was present before, it still is, if it wasn't, is still isn't).

The input is terminated by a description starting with $n = m = 0$. This test case should not be processed.

Output

For each product description first output the number of the product. Then output whether there is a sequence of patches that removes all bugs from a product that has all n bugs. Note that in such a sequence a patch may be used multiple times. If there is such a sequence, output the time taken by the fastest sequence in the format shown in the sample output. If there is no such sequence, output "`Bugs cannot be fixed.`".

Print a blank line after each test case.

Sample Input

```
3 3
1 000 00-
1 00- 0-+
2 0-- -++
4 1
7 0-0+ ----
0 0
```

Sample Output

```
Product 1
Fastest sequence takes 8 seconds.

Product 2
Bugs cannot be fixed.
```