The 2009 ACM Asia Programming Contest Wuhan Site
sponsored by IBM
hosted by Wuhan University

Problem A
Assembling Services
Input: assembling.in

In this problem, you need to simulate the execution of n service programs P_1, P_2, ..., P_n. Each program is described with a sequence of integers: T I in_1 in_2 ... in_I O out_1 out_2 ... out_O, that means it takes T unit time to execute, needs I input variables (i.e. in_1 in_2 ... in_I), and sets O output variables (i.e. out_1 out_2 ... out_O) when it finishes running. A program can be started if and only if all these T input variables are ready (initially available, or set by some other programs).

Imagine you have a super-computer which can execute as many programs in parallel as you like, and every variable can be read and written simultaneously by multiple programs. Your task is to calculate a particular "target" variable, as soon as possible.

Assume there are 4 programs, shown in the table below:

Program No.	Time	Requires	Produces
1	2	X_1	X_2
2	3	X_1	X_3
3	4	X_2	X_4
4	1	X_3, X_4	X_5

The quickest time to get X_5 is 7, if only X_1 is available at startup.

You also need to construct an expression that shows **how to execute the programs to achieve the minimal time**. The grammar of the expression is recursive:

- **Single Program**: `Px`, where $1 \le x \le n$. (i.e. P2, P499, etc). Meaning: execute the program immediately. Then end of this program marks the end of this expression.
- **Execute in serial**: `(S1S2...Sk)`, where every `Si` is an expression. Note that the outermost pair of parentheses is mandatory. Meaning: execute expression `S1`, then `S2` immediately after `S1` ends, then `S3` immediately after `S2` ends, ..., and finally `Sk` immediately after `Sk-1` ends. Then end of expression `Sk` marks the end of the whole expression.
- **Execute in parallel**: `(S1|S2|...|Sk)`, where every `Si` is an expression. Note that the outermost pair of parentheses is mandatory. Meaning: execute expressions `S1`, `S2`, ..., and `Sk` simultaneously. The end of last finished expression marks the end of the whole expression.

One of the possible expressions for the example above is `(((P1P3)|P2)P4)`. `(P1P2P3P4)`is not acceptable, since X_5 is available at time 10 in that expression, later than the optimal time 7.

Input

There will be at most 100 test cases. Each case begins with three integers n, m, o($1 \le n,m \le 500$, $1 \le o \le m$). The number of programs is n, the number of variables is m, and the target variable is X_o. Variables are numbered 1 to m, programs are numbered 1 to n. The next line contains a 01 string of m characters. The i-th character is 1 if and only if the i-th variable is initially available. The target variable is guaranteed to be unavailable at startup. The following n lines describe the programs. Each line begins with an integer T($1 \le T \le 100$), the execution time, and an integer I followed by I integers in_1, in_2, ..., in_I, as stated above, then an integer O followed by O integers out_1, out_2, ..., out_O. $1 \le in_i, out_i <= m$, $1 \le I, O \le 10$. The last test case is followed by $n=m=o=0$, which should not be processed.

Output

For each test case, print the case number and the total time needed to get the target variable. If it's not possible to get the target variable, print `-1` in stead.

If it's possible to get the target variable, print the expression after that, in the same line. Be sure to print a valid expression having at most 10,000 characters, with each program printed at most once. There should be no whitespace characters within the expression.

To make this problem a little bit easier, it's allowed that some programs finish **after** the optimal time, as long as the target variable is available at the optimal time. You're also allowed to print redundant parentheses (pay attention to the expression length, though). If such an expression does not exist, print "`Can't do in serial-parallel.`", without quotes.

Print a blank line after the output of each test case.

Sample Input	Output for the Sample Input
4 5 5 10000 2 1 1 1 2 3 1 1 1 3 4 1 2 1 4 1 2 3 4 1 5 1 2 1 01 31 1 2 1 1 3 5 5 10100 3 1 1 1 2 1 1 3 1 4 3 2 4 2 1 5 1 3 3 100 1 1 1 1 2 0 0 0	Case 1: 7 (((P1P3)|P2)P4) Case 2: 31 P1 Case 3: 6 ((P1P3)|P2) Case 4: -1

Explanation

After a variable is set, it'll keep available forever. That's why `P3` can be executed, in the third example.

Also note that there are some other correct expressions for the first sample, e.g.`((P1P3P4)|P2)`. You can even print `(((P1P3)P4)|P2)` or `((P1(P3P4))|P2)`. Any one of them is acceptable in this problem.