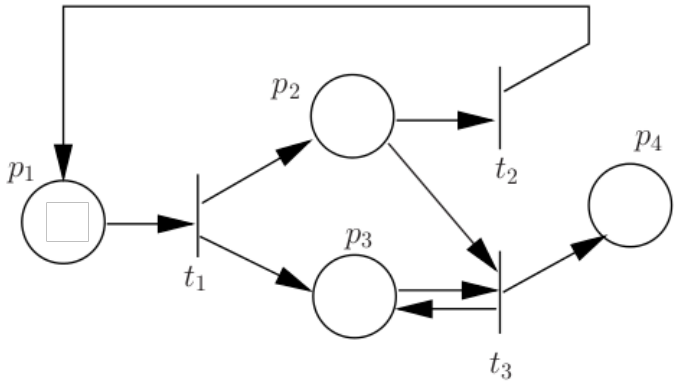
Problem D - Petri Nets

Description

A Petri net is a language to define distributed systems. It can be represented as a directed bipartite multigraph, where each node is either a transition (represented by a bar) or a place (represented by a circle). An arc can only connect a place to a transition or a transition to a place and there can be more than one arc connecting the same pair of nodes. Let *I(t)* and *O(t)* denote the set of places that have ingoing arcs to and outgoing arcs from transition *t*, respectively. The figure below shows an example of a Petri net with 4 places and 3 transitions (taken from [1]).



A state of a Petri net is characterized by a given number of tokens at each place. For instance, the state *[1,0,0,0]* in the Petri net above means that there exists one token in place p_1 and no token in the remaining places.

Let *x(s, p)* be the number of tokens at place *p* in state *s*. Given a state *s*, to fire a transition *t* means to perform the following operation: for each place *p* in *I(t)*

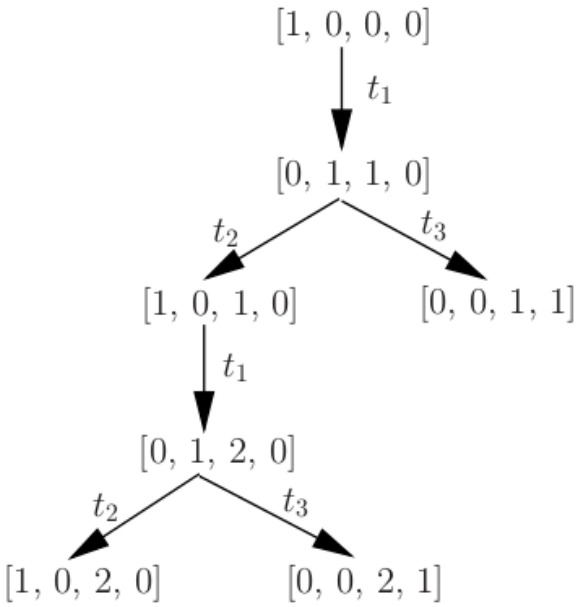
- *x(s,p) = x(s,p) - 1, for each arc (p,t)*

and for each place *p* in *O(t)*

- *x(s,p) = x(s,p) + 1, for each arc (t,p)*

A transition *t* can only be fired if for all places *p* in *I(t)*, *x(s,p)* is equal to or larger than the number of arcs *(p,t)*. If, in a given state, it is not possible to fire any transition, then we say that a *deadlock* occured.

The goal is to print all states that can be reached from an initial state, by considering all possibilities of firing operations. This generation process can be represented as a tree, namely, a *reachability tree*. The initial reachability tree for the Petri net above is shown below (using a DFS strategy) [1].

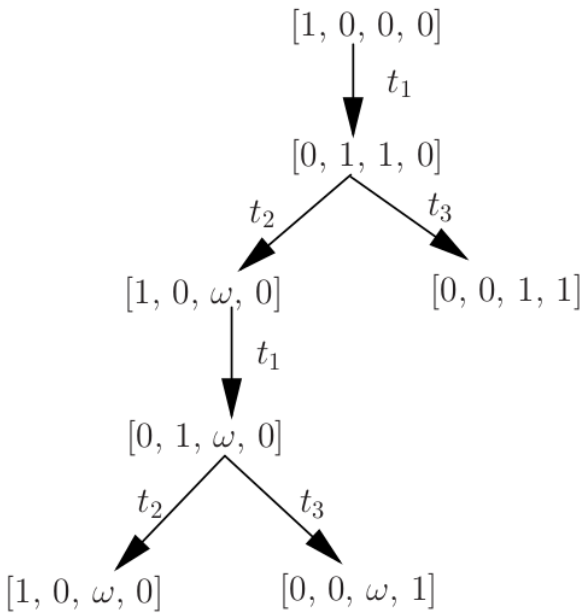


Note that is not possible to expand state *[0,0,1,1]* as well as state *[0,0,2,1]* since they correspond to a deadlock. Also, if a given state *s* arises again in the path from *s* to root, there is no need to expand it. However, the reachability tree may be infinite. For instance, in the example above, there is a repeating pattern of *[1,0,*,0]* after t_2 and *[0,1,*,0]* after t_1 in the left branch, which suggests that the reachability tree may be infinite.

We impose an additional condition at each node of the reachability tree in order to bound its size. We say that a state s *dominates* a state *r*, if

1. $x(r,p) \leq x(s,p)$ for all places *p*
2. $x(r,p) < x(s,p)$ for at least one place

If a new generated state *s* dominates a predecessor state *r* in the path to the root, insert symbol *w* in state *s* where $x(r,p) < x(s,p)$ occurs instead of the number of tokens. From now on, all successor states of state *s* will have a *w* in that position. The transformed tree is called *coverability tree* and it has been shown that this tree is always finite (see references in [1]). The tree below represents the coverability tree of the example above. Note that there is no need to expand state *[1,0,w,0]* since it is a duplicated state.



Input

Each test case starts with two positive integers in the same row, the number of places (n) and the number of transitions. Then, several lines follow, each of which gives information about an arc in the Petri net. Each arc is identified by two positive integers which correspond to the id of the transition and the id of the place. A third positive integer, which takes only number 1 or 2, provides information about which of the two previous numbers corresponds to the transition. The list of arcs ends with the string "STATE", which is followed by a line with n non-negative values, corresponding to the initial state.

Output

For each test case, print the coverability tree starting from the initial state given in the input and using a DFS strategy. When branching, consider always the numerical ordering of the transitions. Each line should consist of n non-negative integers corresponding to a given state, preceded by the number of spaces equals to the height of the tree minus one, at that state.

Constraints

- $n \leq 5$
- $m \leq 5$

IMPORTANT NOTE: You are not allowed to use global variables or macros. Every submission that does not follow this rule will not be considered for

assessment.

Example

Example input:

```
4 3
1 1 2
1 2 1
1 3 1
2 2 2
2 1 1
2 3 2
3 3 1
3 3 2
3 4 1
STATE
1 0 0 0
```

Example output:

```
1 0 0 0
 0 1 1 0
  1 0 w 0
   0 1 w 0
    1 0 w 0
    0 0 w 1
  0 0 1 1
```

References

[1] C.G.Cassandras, S. Fartoune, Introduction to Discrete Event Systems, Springer, 2008.