Problem C - Analyzing a Data Pipeline

Description

As part of your job, you have been asked to design a data pipeline that loads a data set and performs a series of processing operations on it, until you get a final result that should be stored somewhere, e.g., in a file or database.

For analysis purposes, you represent the data pipeline you are developing as a network of operations, such that a connection from one operation to another, means that the latter depends on the former being completed first. This means, that one operation can only start once all its dependencies have finished. Moreover, you have determined that each operation takes a given amount of time for a particular data set.

You want to develop a program to analyze the data pipeline in several ways. First of all, you want to make sure that the data pipeline is valid. This means that: the whole network should be connected; there should be a single initial node that the whole network diverges from (which corresponds to the operation of loading the data set); there should be a single terminal node that the whole network converges to (which corresponds to the operation of storing the result somewhere); and there should be no cycles, since otherwise the pipeline can never finish.

If the network is valid, then you want to compute several statistics about the data pipeline.

1. Assuming that you would run this pipeline in a computer that can only perform one operation at a time, what is the minimum amount of time needed to run the pipeline, and what is a feasible order to run the operations?

2. Assuming that you would run this pipeline in a computer that can run an infinite number of operations in parallel, what is the minimum amount of time needed to run the pipeline?

3. Which operations are a bottleneck for the pipeline? An operation is considered a bottleneck, if it cannot be run in parallel with others.

The program you are developing to analyze the data pipeline will receive the network representing it, and be asked to compute one particular statistic.

Input

The first line of the input contains a positive integer N that gives the number of operations. Operations are numbered from 1 to N. Note that, operations are numbered arbitrarily, as such, operations 1 and N are **not** necessarily the initial and terminal operations of the data pipeline.

The next N lines describe the operations, such that the first line describes operation 1, the second line operation 2, and so on. Each line, starts with two positive integers T and D. The first corresponds to the amount of time needed to process the operation. The second gives the number of dependencies for this operation. Then, D integers follow, denoting which operations need to be done before this one.

The last line of the input gives an integer 0, 1, 2, or 3, which denotes the statistic that should be computed if the data pipeline is valid. If 0 it should only check if the data pipeline is valid, otherwise it should compute the corresponding statistic enumerated in the description.

Output

If the pipeline is invalid, the program should simply print `INVALID`. Otherwise, its output depends on the statistic:

- For statistic 0, it should print `VALID`.
- For statistic 1, it should first print an integer denoting the minimum amount of time the data pipeline takes to process if only one operation can be processed at a time. Then, in the following N lines, it should print a feasible order for running the operations, one per line. In case there is more than one operation that can run at any given time, the numerically smallest should be printed.
- For statistic 2, it should output the minimum amount of time the pipeline takes to process if an infinite number of operations can be processed simultaneously.
- For statistic 3, it should print the ids of the operations that are bottlenecks, one per line. These should be printed in the order that they are processed in the data pipeline. Note that, per definition, the initial and terminal operations are always bottlenecks.

Limits

- $3 \leq N \leq 1000$
- $1 \leq T \leq 100$

Example 1

Input

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

Output

```
INVALID
```

Explanation

Node 4 is disconnected from nodes 1, 2, and 3.

Example 2

Input

```
5
3 0
2 1 1
2 0
5 2 2 3
1 1 4
1
```

Output

```
INVALID
```

Explanation

There are two possible initial nodes, 1 and 3.

Example 3

Input

```
6
3 0
2 1 1
2 1 1
5 2 2 3
1 1 4
2 1 4
2
```

Output

```
INVALID
```

Explanation

There are two possible terminal nodes, 5 and 6.

Example 4

Input

```
5
3 0
2 2 1 4
2 2 1 2
5 1 3
1 1 4
3
```

Output

```
INVALID
```

Explanation

There is a cycle between nodes 2, 3, and 4.

Example 5

Input

```
5
3 0
2 1 1
2 1 1
5 2 2 3
1 1 4
0
```

Output

```
VALID
```

Example 6

Input

```
5
3 0
2 1 1
2 1 1
5 2 2 3
1 1 4
1
```

Output

```
13
1
2
3
4
5
```

Explanation

The second operation to run could either be 2 or 3. However, the output description asks us to print the numerically smallest.

Example 7

Input

```
5
3 0
2 1 1
2 1 1
5 2 2 3
1 1 4
2
```

Output

```
11
```

Example 8

Input

```
5
2 1 2
3 0
2 1 2
1 1 5
5 2 1 3
3
```

Output

```
2
5
4
```