CMPUT 275 - Tangible Computing
Interview Problem: Makefile Length

Comment: This interview problem was given to one of our TAs at their interview for an internship at IBM. That is, the algorithmic challenge was the same: we just dressed it up with a different story based on Makefiles.

Description

In this problem, you will first be given a list of `Makefile` targets and their dependencies (the actual build commands, eg. `g++` commands, do not matter for this exercise). Such a target will appear like:

```
<targetname>:  <list of dependencies>
```

Here, the list of dependencies will be a space-separated list of names. Some files listed as a dependency to a target may themselves be targets.

You are tasked with answering the following question: what is the length of the longest sequence of dependencies of the form `A depends on B depends on C depends on...`?

More precisely, what is the largest ℓ such that there is a sequence of the form $f_0, f_1, \ldots, f_\ell$ where each f_i is a file name mentioned in the `Makefile` and for $1 \leq i \leq \ell$, f_i is listed as one of the dependencies of f_{i-1}.

Input

The first line of input a single integer $1 \leq n \leq 100,000$, the amount of targets in the makefile. Then n lines follow, each describing a target and its dependencies. Each such line will take the form:

```
k <targetname>:  <list of dependencies>
```

where $k \geq 1$ is an integer specifying the number of dependencies of the target. Exactly one space will separate the different dependencies in the list and there is exactly one space between the colon character following the target name and the first dependency. There is no space separating the target name and the colon.

You are given the following additional guarantees:

- The total number of all dependencies of all targets is at most 300,000 (i.e. the sum of all the k values on n target lines is at most 300,000).
- All file names consist of letters, digits, or the dot symbol (a period). No file name contains any space. All file names will have at most 12 characters.
- Different files will not have the same name. So if you see a file name listed twice among the targets and dependencies, it is the same file.
- No file will be listed as a target more than once.
- There will be no "cycle" of dependencies. For example, if file1 depends on file2 and file2 depends on file 3, then file3 will not depend on file1. More generally, if we consider

the directed graph over files where we include a directed edge from X to Y if Y is listed as a dependency of X, then the graph will not contain any cycles. This also means a file will not depend on itself.

- Any file name listed as a dependency but not as a target is a file given by the user.

Output

You are to output one line containing a single integer that answers the question: what is the longest sequence of dependencies of the form `A depends on B depends on C depends on...`?

Running Time

There is no specific $O()$ running time you must achieve on this weekly exercise. The ideal running time is linear in the total size of the input (i.e. the number of targets + total number of dependencies). It will not be possible for a significantly slower algorithm to pass the larger test cases.

Dynamic Programming

Use dynamic programming to solve this problem. This algorithm design paradigm will be discussed in the lecture on Thursday, April 4. **Helpful tip**: It will be useful to compute the following for *every* file name `fname`: what is the length of the longest sequence of dependencies that begins with `fname`? There is a natural top-down dynamic programming approach to computing these values.

Graph Class + Other Requirements

You are **not** required to use the graph class developed in the lectures. However, if you wish to do so to model the dependency relationships in this problem, you should put the class declaration and method implementations in the same source code file you use to solve this problem. This is because you are to submit only a single source code file (we will not regard this as bad style). You may make any modifications you wish to the graph class.

You may include any other libraries or create any helper functions you wish to solve this problem. There are no strict requirements on the names of the functions you use.

Sample Input 1

```
5
3 project: main.o matrix.o fft.o
3 debug: main.o matrix.o fft.o
3 main.o: main.cpp matrix.h fft.h
2 matrix.o: matrix.h matrix.cpp
2 fft.o: fft.h fft.cpp
```

Sample Output 1

```
2
```

Explanation
There are many such sequences, one is `project` depends on `main.o` depends on `main.cpp`

Sample Input 2

```
8
3 a: b c g
3 b: e f g
3 c: b e d
1 d: f
1 g: e
3 sss: xxx yyy zzz
2 xxx: yyy zzz
1 yyy: zzz
```

Sample Output 2

```
4
```

Explanation
The only sequence of dependencies this long is:
`a` depends on `c` depends on `b` depends on `g` depends on `e`

Grading Comments
Despite the fact this appears similar to a morning problem, it will be graded like a weekly exercise. In particular:

- Style matters. Use appropriate comments, proper indentation, etc. Include a file header. Consult the style guide on eClass. **Recall**: We are going allowing you to include the class declaration and method implementations of a graph class (or any other class you deem useful) in the file.
- You must adhere exactly to the output specification: for example, if you output the lines in the wrong order or print extra whitespaces then you will receive a deduction. For full credit, the test centre must accept the output without any presentation error.
- You were only give a few test cases in the test centre files on eClass. We will test your solution on additional test cases that adhere to the input specification.
- Partial credit may be obtained if your solution works on some inputs (eg. small instances).
- Adhere closely to the submission instructions for the weekly exercise. See the eClass code submission link for details.