A client wants us analyze a set of documents by computing statistics on how each document has changed over time. They've provided us with a zip archive containing all of the versions they have of each document. Unfortunately, all of the files are intermingled, and the files themselves do not identify which document they are a version of.

Therefore, before we can perform the analysis, we must attempt to partition the files such that all files in the same group are thought to be versions of the same document.

The client is vaguely optimistic that this is possible. They warn that there is certain content that is expected to appear in all documents, but they feel that the amount of this shared content should be low enough that adjacent versions of the same document will always have more in common than any two versions of different documents.

The archive files.zip contains 20,000 text files named with successive integers: 1.txt, 2.txt, …, 19999.txt. The first line of each file is an integer timestamp that identifies when that version was published. (Apparently, no two versions of the same document will have the same timestamp.) All subsequent lines in the file are considered content and are composed of words separated with spaces.

Between adjacent versions of the same document, content lines may have been added, removed, or edited. However, as noted above, it is believed that such changes will never be pervasive enough to prevent associating the versions.

No matter what the client says, we are skeptical that a perfect classification is feasible for such a large data set, and we have refused to guarantee one. However, we feel it should be possible to get reasonably close.

Please write a program that outputs a proposed partition for the files. For each proposed grouping, print the sorted file numbers, delimited by spaces, on a separate line. Sort the lines by the lowest number in the line. For example, if you believe 11.txt, 7.txt, and 99.txt belong to one document, and 12.txt and 3.txt to another, the output would be

3 12

7 11 99

Notes:

This is a simplified version of a real client problem Steelray worked on in January 2024. As with the real version, the goal isn't necessarily to produce the "right answer" (if one even exists), but to get something that is good enough, and explain whatever shortcuts or compromises were needed to get there.

This is a new interview problem for Steelray, meaning that it's hard for us to gauge how difficult it might be given specific time constraints. Feel free to adjust the problem, if needed, to make it more reasonable.

For example, the supplied data set is large enough to offer plenty of scope for optimization. However, if you don't have time for much optimization, or decide to focus on other aspects, it might take forever for

your algorithm to chew through the entire data set. In that case, you might opt to only analyze a subset of the files.

Ultimately, this problem is a means to have an interesting technical conversation, hopefully resembling one we might have as coworkers working on a real project. When in doubt, optimize for the interestingness of that conversation, rather than any fixed technical criterion.