Problem D: I Conduit!

Irv Kenneth Diggit works for a company that excavates trenches, digs holes and generally tears up people's yards. Irv's job is to make sure that no underground pipe or cable is underneath where excavation is planned. He has several different maps, one for each utility company, showing where their conduits lie, and he needs to draw one large, consolidated map combining them all. One approach would be to simply draw each of the smaller maps one at a time onto the large map. However, this often wastes time, not to mention ink for the pen-plotter in the office, since in many cases portions of the conduits overlap with each other (albeit at different depths underground). What Irv wants is a way to determine the minimum number of line segments to draw given all the line segments from the separate maps.

Input

Input will consist of multiple input sets. Each set will start with a single line containing a positive integer n indicating the total number of line segments from all the smaller maps. Each of the next n lines will contain a description of one segment in the format

$x_1\ y_1\ x_2\ y_2$

where (x_1, y_1) are the coordinates of one endpoint and (x_2, y_2) are the coordinates of the other. Coordinate values are floating point values in the range $0 \ldots 1000$ specified to at most two decimal places. The maximum number of line segments will be 10000 and all segments will have non-zero length. Following the last input set there will be a line containing a 0 indicating end of input; it should not be processed.

Output

For each input set, output on a single line the minimum number of line segments that need to be drawn on the larger, consolidated map.

Sample Input

```
3
1.0 10.0 3.0 14.0
0.0 0.0 20.0 20.0
10.0 28.0 2.0 12.0
2
0.0 0.0 1.0 1.0
1.0 1.0 2.15 2.15
2
0.0 0.0 1.0 1.0
1.0 1.0 2.15 2.16
0
```

Sample Output

```
2
1
2
```

Problem E: Roll Playing Games

Phil Kropotnik is a game maker, and one common problem he runs into is determining the set of dice to use in a game. In many current games, non-traditional dice are often required, that is, dice with more or fewer sides than the traditional 6-sided cube. Typically, Phil will pick random values for all but the last die, then try to determine specific values to put on the last die so that certain sums can be rolled with certain probabilities (actually, instead of dealing with probabilities, Phil just deals with the total number of different ways a given sum can be obtained by rolling all the dice). Currently he makes this determination by hand, but needless to say he would love to see this process automated. That is your task.

For example, suppose Phil starts with a 4-sided die with face values 1, 10, 15, and 20 and he wishes to determine how to label a 5-sided die so that there are a) 3 ways to obtain a sum of 2, b) 1 way to obtain a sum of 3, c) 3 ways to obtain 11, d) 4 ways to obtain 16, and e)1 way to obtain 26. To get these results he should label the faces of his 5-sided die with the values 1, 1, 1, 2, and 6. (For instance, the sum 16 may be obtained as $10 + 6$ or as $15 + 1$, with three different "1" faces to choose from on the second die, for a total of 4 different ways.)

Input

Input will consist of multiple input sets. Each input set will start with a single line containing an integer n indicating the number of dice that are already specified. Each of the next n lines describes one of these dice. Each of these lines will start with an integer f (indicating the number of faces on the die) followed by f integers indicating the value of each face. The last line of each problem instance will have the form

$$r \ m \ v_1 \ c_1 \ v_2 \ c_2 \ v_3 \ c_3 \ \cdots \ v_m \ c_m$$

where r is the number of faces required on the unspecified die, m is the number of sums of interest, $v_1, \ldots, v_m$ are these sums, and $c_1, \ldots, c_m$ are the counts of the desired number of different ways in which to achieve each of the respective sums.

Input values will satisfy the following constraints: $1 \leq n \leq 20$, $3 \leq f \leq 20$, $1 \leq m \leq 10$, and $4 \leq r \leq 6$. Values on the faces of all dice, both the specified ones and the unknown die, will be integers in the range $1 \ldots 50$, and values for the v_i's and c_i's are all non-negative and are strictly less than the maximum value of a 32-bit signed integer.

The last input set is followed by a line containing a single 0; it should not be processed.

Output

For each input set, output a single line containing either the phrase "`Final die face values are`" followed by the r face values in non-descending order, or the phrase "`Impossible`" if no die can be found meeting the specifications of the problem. If there are multiple dice which will solve the problem, choose the one whose lowest face value is the smallest; if there is still a tie, choose the one whose second-lowest face value is smallest, etc.

Sample Input

```
1
4 1 10 15 20
5 5 2 3 3 1 11 3 16 4 26 1
1
6 1 2 3 4 5 6
6 3 7 6 2 1 13 1
4
6 1 2 3 4 5 6
4 1 2 2 3
3 3 7 9
8 1 4 5 9 23 24 30 38
4 4 48 57 51 37 56 31 63 11
0
```

Sample Output

```
Final die face values are 1 1 1 2 6
Impossible
Final die face values are 3 7 9 9
```

Problem H: Translations

Bob Roberts is in charge of performing translations of documents between various languages. To aid him in this endeavor his bosses have provided him with translation files. These files come in twos — one containing sample phrases in one of the languages and the other containing their translations into the other language. However, some over-zealous underling, attempting to curry favor with the higher-ups with his initiative, decided to alphabetically sort the contents of all of the files, losing the connections between the phrases and their translations. Fortunately, the lists are comprehensive enough that the original translations can be reconstructed from these sorted lists. Bob has found this is most usually the case when the phrases all consist of two words. For example, given the following two lists:

Language 1 Phrases	**Language 2 Phrases**
arlo zym	*bus seat*
flub pleve	*bus stop*
pleve dourm	*hot seat*
pleve zym	*school bus*

Bob is able to determine that *arlo* means *hot*, *zym* means *seat*, *flub* means *school*, *pleve* means *bus*, and *dourm* means *stop*. After doing several of these reconstructions by hand, Bob has decided to automate the process. And if Bob can do it, then so can you.

Input

Input will consist of multiple input sets. Each input set starts with a positive integer n, $n \leq 250$, indicating the number of two-word phrases in each language. This is followed by $2n$ lines, each containing one two-word phrase: the first n lines are an alphabetical list of phrases in the first language, and the remaining n lines are an alphabetical list of their translations into the second language. Only upper and lower case alphabetic characters are used in the words. No input set will involve more than 25 distinct words. No word appears as the first word in more than 10 phrases for any given language; likewise, no word appears as the last word in more than 10 phrases. A line containing a single 0 follows the last problem instance, indicating end of input.

Output

For each input set, output lines of the form

word1/word2

where *word1* is a word in the first language and *word2* is the translation of *word1* into the second language, and a slash separates the two. The output lines should be sorted according to the first language words, and every first language word should occur exactly once. There should be no white space in the output, apart from a single blank line separating the outputs from different input sets. Imitate the format of the sample output, below. There is guaranteed to be a unique correct translation corresponding to each input instance.

Sample Input

```
4
arlo zym
flub pleve
pleve dourm
pleve zym
bus seat
bus stop
hot seat
school bus
2
iv otas
otas re
ec t
eg ec
0
```

Sample Output

```
arlo/hot
dourm/stop
flub/school
pleve/bus
zym/seat

iv/eg
otas/ec
re/t
```