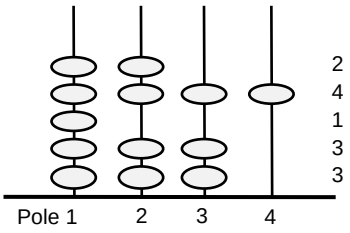
ACM Southeast Regional
Intercollegiate Programming Contest
Saturday, 13 October 2007

Georgia Southern University
University of South Alabama
Florida Institute of Technology

Problem Set

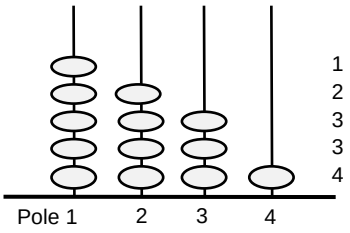
1. Abacus Sort

In this problem you'll have to implement the Abacus Sort algorithm, also known as the Bead Sort algorithm.



This abacus includes N poles, where N is the largest number to be sorted. Unlike conventional abacuses, a number is represented by a corresponding number of beads on a separate row, starting at the left (pole 1), and moving towards the right.

Once all numbers have been placed on their own row, we tilt the abacus and let the beads fall. In this case, four beads move downward.



At this point, the top row of the abacus will represent the smallest number, and we can read off all the numbers, in sorted order, by reading the values on the various rows, from the top down.

Input

The input file contains one or more test cases. Each test case consists of n lines ($0 < n \leq 80$), each line with an integer, val ($0 < val \leq 50$), to be sorted. After each test case, there will be a line with a single 0. All integers will begin in the first column of the line. When reading input, the first item read will be the item at the top of the abacus when it is tilted and the last item read will be the item at the bottom.

The end of input will be indicated by end of file.

Output

For each test case in the input file, you should produce the listing of the sorted values in ascending order. Each value should be displayed in its own row of text, with an X representing a pole with a bead and a hyphen representing a pole without a bead. There should be exactly N poles represented for each integer, where N is the largest number to be sorted in the set. At the end of the row, have a blank, a greater than ('>') sign, a blank, and the integer being represented.

At the end of the final depiction of the abacus, have a line telling how many beads moved as a result of doing the sort. For example, in the example above (the first example in the sample input), 4 beads move. Your statement should have the form:

`A total of xx beads slid during the sort.`

where *xx* is replaced by the number of beads that move. You should not change "beads" to "bead" in the case of 1 bead sliding.

Have a blank line between output from each input set.

Sample Input

```
2
4
1
3
3
0
30
40
40
20
0
```

Output Corresponding to Sample Input

```
X--- > 1
XX-- > 2
XXX- > 3
XXX- > 3
XXXX > 4
A total of 4 beads slid during the sort.

XXXXXXXXXXXXXXXXXXX-------------------- > 20
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX---------- > 30
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX > 40
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX > 40
A total of 50 beads slid during the sort.
```

2. Base-*d* Times

Your little sister Annie has just learned about different number bases. When you ask her what time it is, she tells you it's 5. "Five?" you ask. "Yes," she explains, since the clock shows "1:01" which, she proudly explains, "is 5 in base-2." Since you have to live with your sister and know she's going to learn more bases, you decide to write a program to make sense of her answers.

Your program should find all the times in an inclusive range that could be considered to valid numbers in a given base (ignoring the colon). For example, in base-2, in the inclusive time range between 1:00 and 2:00, the times 1:00, 1:01, 1:10, and 1:11 can be interpreted as the numbers 100, 101, 110, and 111 in base-2 (i.e. 4, 5, 6, and 7 in decimal).

Input

The input to the program will consist of one or more data sets. Each data set will be on a single line. It will begin with the base, d ($1 < d \leq 10$), to use in examining clock times. There will then be 2 times, the start time and the end time, each in the format *hh*:*mm* or *h*:*mm*, with $0 \leq h \leq 9$, $10 \leq hh \leq 23$, and $0 \leq mm \leq 59$. There will be one or more spaces between these values. All times are assumed to be in the same day and the end time will be later than the start time.

The last data set will consist of just the value 0. This line should not be processed.

Output

For each data set, have an initial line giving the range of times, in the format shown in the Sample Output section. Then have a list of all decimal integers that correspond to times in the inclusive range that are valid base-*d* numbers. These numbers should be listed in increasing order, one to a line, and indented by three spaces. Have one blank line after the output for each data set.

Sample Input

```
2 1:00 4:00
3 1:15 1:45
0
```

Output Corresponding to Sample Input

```
1:00 to 4:00:
   4
   5
   6
   7

1:15 to 1:45:
   15
   16
   17
```

3. The Dog Days of Programming Contests

Dr. Hain and Dr. White have decided to retire from judging the regional programming contest and instead want to become dog show judges. Unfortunately, dog shows use different criteria from programming contests (dogs very seldom have compiler errors, although some may have wrong output), so they need a way to ensure they are judging the dogs consistently. Since they are still the chief judges of this contest, they have asked you to write a program to check the consistency of their dog judging.

The Problem

Your program should create charts showing the consistency between the two judges. The rows will represent the scores given by Dr. Hain and the columns will represent the scores given by Dr. White. Suppose the maximum score for a dog is 3 and the judges give the following scores:

Dog number	Dr. Hain's score	Dr. White's score
1	0	0
23	0	3
14	0	0
5	3	0
19	2	3
18	1	1
12	3	2
204	1	1
6	0	0
25	3	3

The consistency chart resulting from these scores is:

		Dr. White's Score			
		0	1	2	3
Dr. Hain's Score	0	3	0	0	1
	1	0	2	0	0
	2	0	0	0	1
	3	1	0	1	1

The 3 in the upper left hand corner indicates there were 3 dogs that Dr. Hain gave a 0 to that Dr. White also gave a 0 to (dogs 1, 14, and 6). The 1 in row 0, column 1 indicates there was 1 dog that Dr. Hain scored as a 0 while Dr. White scored it as a 3 (dog 23). Ideally, the only nonzero entries in this table would be along the diagonal, indicating perfect consistency.

Your program should take a group of scores and build the corresponding consistency table.

Input

The input to the program will consist of one or more data sets. Each data set will begin with a single integer, *max*, $0 < max \leq 20$, the maximum score a dog may receive. There will then be 1 or more lines representing the scores given by the judges. They will be in the following format:

judge dog score

where:

- *judge* is a single letter, 'H', 'W', or 'E' with 'H' representing Dr. Hain's judgment, 'W' representing Dr. White's judgment, and 'E' marking the end of scores for this data set.
- *dog* is a positive integer representing the dog number.
- *score* is an integer, $0 \leq score \leq max$

The end of input will be indicated by data set with max ≤ 0. This set should not be processed.

Output

The output for each data set should start with a single line indicating the set number, starting at 1 and formatted as in the sample output. The consistency matrix should then be printed in row major order, one row per line, with one blank between each pair of consecutive entries. Only dogs which have been scored by both judges should be included in the consistency matrix. There should be a blank line after the consistency matrix for each data set.

Sample Input

```
3
H 1 0
H 19 2
W 25 3
H 12 3
H 204 1
H 6 0
W 1 0
H 5 3
H 25 3
W 23 3
W 14 0
W 5 0
W 19 3
H 23 0
W 18 1
H 14 0
H 18 1
W 12 2
W 204 1
W 6 0
H 10 2
W 100 2
E 0 0
2
H 1000 1
W 1000 0
H 1001 1
W 1001 0
H 99 1
H 5 1
H 1002 1
W 1002 0
H 1003 1
W 1003 0
H 1004 1
W 1004 0
H 1005 1
W 1005 0
H 1007 1
W 1007 0
H 1008 1
W 1008 0
H 1009 1
W 1009 0
E 1 1
-55
```

Output Corresponding to Sample Input

```
Show 1:
3 0 0 1
0 2 0 0
0 0 0 1
1 0 1 1

Show 2:
0 0 0
9 0 0
0 0 0
```

4. How Do I Get Back HOME?

I drove all the way from my house to the contest site, so that I could carry out my divine duty as a contest judge. I printed out directions from my house to the contest, but forgot to print directions from the contest location back to my house! Your job is to generate (from the directions to the contest) the directions to get me back home.

The directions data has the following grammar:

<directions> = <start-instruction>   <traveling-instruction-list>   <end-instruction>
<start-instruction> = `Head` <compass> `from` <address>
<end-instruction> = `Arrive at` <address>
<address> = <number> <road>
< traveling-instruction-list > = <instruction>
< traveling-instruction-list > = <instruction>   <instruction-list>
<instruction> = `Turn` <LR> `at` <road>
<instruction> = `Continue on` <road>
<road> = <word-list>
<word-list> = <word>
<word-list> = <word> <word-list>
<LR> = `left`
<LR> = `right`
<compass> = `east`
<compass> = `west`
<compass> = `north`
<compass> = `south`
A <number> is an integer from 1 to 999999. A <word> is a string of letters and digits. A single space will separate words and numbers on a line.

You may assume all roads are two-way, and have either a north-south, or an east-west orientation. An instruction reading "Continue on Elm St" means the road has changed its name to Elm St, but there is no change in the direction of travel.

Input

The input will consist of one or more data sets. Each directions data set starts with a line containing one integer, n ($2 < n < 100$ or $n=0$), being the total number of instructions. The next n lines each contain one instruction, and follow the grammar given above. An instruction line has less than 120 characters. The last directions data set has $n=0$, and should not be processed.

Output

The output for each directions data set should begin with a line looking like

`Directions <d>:`

where <*d*> is the number of the data set, starting at 1. There should follow *n* output lines, each being one instruction. There should be a blank line after each data set.

Sample Input

```
7
Head north from 2357 Georgia Ave
Turn right at Bland Ave
Turn right at W Gentilly Rd
Turn left at GA67
Continue on Jones Lane Memorial Hwy
Turn right at E Main St
Arrive at 1 E Main St
8
Head east from 7532 Monroe St
Turn left at Everett St
Turn right at Government St
Continue on US98
Turn left at S Conception St
Continue on N Conception St
Turn right at Saint Louis St
Arrive at 154 Saint Louis St
0
```

Output Corresponding to Sample Input

```
Directions 1:
Head north from 1 E Main St
Turn left at Jones Lane Memorial Hwy
Continue on GA67
Turn right at W Gentilly Rd
Turn left at Bland Ave
Turn left at Georgia Ave
Arrive at 2357 Georgia Ave

Directions 2:
Head west from 154 Saint Louis St
Turn left at N Conception St
Continue on S Conception St
Turn right at US98
Continue on Government St
Turn left at Everett St
Turn right at Monroe St
Arrive at 7532 Monroe St
```

5. Modified Run Length Encoding

Run length encoding (RLE) of text strings replaces sequences of characters by a number (the number of repetitions of the character) followed by the character that is repeated. Thus, the string AAAAABAAABBCEEEEE would be replaced by 4AB3ABBC5E. Note that BB is not replaced by 2B since that is no shorter.

Under some circumstances, modified run length encoding (MRLE) can do better by replacing repeated adjacent substrings in a similar way. We now need a pair of numbers associated with a repeated substring: the number of repetitions, and the length of the substring. The string ABABAB (length 6) can be written[1] as 3 2 AB (length[2] 4), being 3 repetitions of a length-2 substring (AB).

From longest to shortest substring length, the algorithm will scan the target string from left to right looking for sequences of substrings of that length. When a repetition of a substring is found, it is replaced the two numbers associated with the substring, followed by the substring itself. This process is repeated until finally all length-1 substrings have been considered.

For example, the string BABABABAABABABAB is first reduced to B2 7 ABABABAB (the longest repeating substring is ABABABA; the ABABABAABABABA is replaced by 2 7 ABABABA). This reduced string is then further reduced to B2 7 2 4 ABAB (the longest repeating substring after the first reduction is ABAB; the ABABABAB is replaced by 2 4 ABAB). No further reductions in the length of the compressed string can be made (The replacement of ABAB by 2 2 AB does not reduce the length).

Input

Each target string may be spread over several lines. The first line for each target string contains a single integer, (n<100), representing the number of lines containing the sequential components of the target string. This is followed by n lines containing strings of 1 to 256 uppercase characters. The first line of the last target string contains 0, and should not be processed.

The target strings are guaranteed not to have repeated substrings longer than 100, and the number of repetitions is guaranteed to be no more than 100.

---

[1] To be able to unambiguously display sequences of numbers (which may have more than one decimal digit), we will always display a number followed by a space.

[2] For simplicity, we will only consider cases where a number can be encoded in a byte, as well as being distinguishable from characters.

Output

There should be two output lines for each target string. The first line should be

`Target <n>:`

where <*n*> is the number of the target string, starting at 1. The second line should be the compressed string.

Sample Input

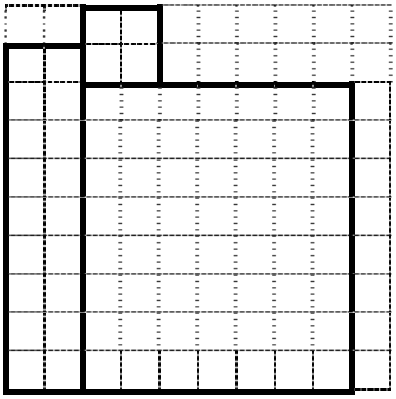
```
1
BABABABAABABABAB
3
ABBCCCD
DDDEF
EFGHGHGHIJIJIJ
8
BBBB
BAAAABAAAABAAAAC
BAAAABAAAABAAAACD
BAAAABAAAABAAAAC
BAAAABAAAABAAAACD
BAAAABAAAABAAAAC
BAAAABAAAABAAAACD
EE
0
```

Output Corresponding to Sample Input

```
Target 1:
B2 7 2 4 ABAB
Target 2:
ABBCCC4 1 DEFEF3 2 GH3 2 IJ
Target 3:
4 1 B3 33 2 16 3 5 B4 1 ACDEE
```

6. Persian Carpets

Jane has just bought three rectangular Persian carpets that she wants to show off in the new house that she's building. She wants to put all the carpets into a square room, such that none of the carpets overlap, and that all carpet edges are parallel to the room walls. Given the sizes of the carpets, she wants to know the dimensions of the smallest room that will hold the carpets. For example, if the carpets are sized 9×2, 2×2, and 8×7, these could be placed into a 10×10 room as follows:



Input

Each data set contains three lines, each containing two positive integers (<1000) separated by a single space, being the dimensions of one carpet. The end of data is indicated with a single line having the values 0 0.

Output

For each data set, output a line looking like

`Set <n>: minimum dimension = <d>`

Where <*n*> is the data set number starting at 1, and <*d*> is the size of the square room.

Sample Input

```
9 2
2 2
8 7
8 3
3 7
6 3
0 0
```

Output Corresponding to Sample Input

```
Set 1: minimum dimension = 10
Set 2: minimum dimension = 9
```

7. The Polyathlon

A decathlon is a sporting event with ten different events, and their results are combined to find an overall winner. In a polyathlon, the contestants participate in a variable number of events. You will write code to determine the order of finish of a group of participants in such a multi-event competition.

Problem Description

There are two types of events: timed events and measured events.

- Timed events will have the contestant scores recorded in one of three forms: *hh*:*mm*:*ss* or *h*:*mm*:*ss* or *mm*:*ss,* where *h* is a 1-digit integer, and *hh*, *mm*, and *ss* are 2-digit integers (representing hours, minutes, and seconds), with $0 \leq mm, ss \leq 59$. Example times are 12:01:01, 3:24:05 and 02:27. Example timed events are a marathon, or time to find a hidden treasure. In a timed event, a lower time is better.

- Measured events will have the contestant scores recorded as integers. Example measured events are the distance a discus is thrown, or the number of hot dogs eaten in 5 minutes. In a measured event, a higher score is better.

After each event, the contestants are placed in order of their finish. The one with the best score is in position 1, the second best score in position 2, etc. In the case of ties, contestants with the same score are all in the same position. But if n contestants are in position p, the next contestant will be in position $p + n$. For example, consider the contestant results and positions in a measured event below:

Position	Score	Name
1	95	ABRAMS
2	83	BLAKE
2	83	CHIN
4	82	DELNOSTRO
5	70	EWING
5	70	FRANKLYN
5	70	GHOST
8	4	HARDY

Since two participants have the second highest score, 83, they are both in position 2. The third highest score is then in position 4. Participants with equal positions are ordered by alphabetically increasing order of their name.

Once all events are completed, the positions of each contestant in each event are summed. The final results will put the contestants in order from lowest to highest based on the sum of their positions. The lowest sum is in position 1. Ties in final results are handled in the same way as in events. If a contestant does not have a score in all events, they are omitted from the final results.

Input

The input to your program will be 1 or more data sets. Each data set will begin with two lines. The first line of the data set will be a line of one or more characters in the set {A-Z, 0-9, _},

giving the name of the competition. The next line will be a positive integer, *n* , being the number of events in the competition. There will then be *n* sets of data about each event.

The data about each event will begin with 3 lines describing the event. The first line of the description will be one or more characters in the set {A-Z, 0-9, _}, giving the name of the event. The second line will be either "TIMED" or "MEASURED", telling whether the event is timed or measured. The third line will be a positive integer, *p*, giving the number of participant scores in the event. There will then be *p* lines, each giving a participant name and score.

Each of the participant lines will have a participant name consisting of one or more upper-case characters, a space, and the participant's score. The score will be in the format described in the problem description.

End of input is indicated by end of file.

Output

For each input set, have a line with the name of the competition, a blank line, and then the output for each of the events in the competition, in the order they appear in the competition followed by the final results.

For each event, the output should be the name of the event on a single line, followed by a single line for each competitor giving their position, their score (output in exactly the same format they were input), and their name, in order of their position. If two competitors finish in the same position, they should be listed alphabetically. There should be a blank line after the output for each event.

After all the output for events, there should be a line saying “OVERALL_RANKINGS”, followed by a single line for each competitor giving their position, the sum of their other positions, and their name. Again, these should be in position order, with ties listed alphabetically. You may assume there will be at least one competitor who participates in all events. There should be two blank lines after the output for each competition.

Sample Input

```
ACM_CONTEST_PREP
2
PROBLEMS_SOLVED
MEASURED
3
DIJKSTRA 3
WIRTH 1
HOARE 3
TIME_TO_SOLVE
TIMED
4
WIRTH 02:12
BOHR 12:01:29
KNUTH 02:12
DIJKSTRA 1:49:05
GREAT_BAKING_CONTEST
1
PIES_BAKED
MEASURED
5
CHERRY 3
APPLE 3
LEMON 3
BLUEBERRY 2
PUMPKIN 4
```

Output Corresponding to Sample Input

```
ACM_CONTEST_PREP

PROBLEMS_SOLVED
1 3 DIJKSTRA
1 3 HOARE
3 1 WIRTH

TIME_TO_SOLVE
1 02:12 KNUTH
1 02:12 WIRTH
3 1:49:05 DIJKSTRA
4 12:01:29 BOHR

OVERALL_RANKINGS
1 4 DIJKSTRA
1 4 WIRTH


GREAT_BAKING_CONTEST

PIES_BAKED
1 4 PUMPKIN
2 3 APPLE
2 3 CHERRY
2 3 LEMON
5 2 BLUEBERRY

OVERALL_RANKINGS
1 1 PUMPKIN
2 2 APPLE
2 2 CHERRY
2 2 LEMON
5 5 BLUEBERRY
```

8. Pronounceable Words

You want to produce flash cards having pronounceable (but perhaps meaningless) words for children to practice their phonetics. Here we define a pronounceable word as one having interleaved consonants and vowels. However, there are also 2-letter compound consonants (e.g., “ch’, “th”, “st”), and 2-letter compound vowels (having a vowel sounds such as “ea”, and “ou”).

Input

The input file contains one or more data sets, each containing two lines. The first line contains an integer v ($0 < v \leq 6$), followed by a space-delimited list of v (single or compound) vowels, taken from the set {'a', 'e', 'i', 'o', 'u'} . The second line contains an integer c ($0 < c \leq 6$), followed by a space-delimited list of c (single or compound) vowels, where (single or compound) consonants are letters other than vowels. All single and compound vowels and consonants are made up of lowercase letters. The data set is guaranteed not to produce more than 10000 distinct pronounceable words.

The end of input will be indicated by a line containing a single 0.

Output

For each data set, produce two lines looking like

```
Data set n:
m words.
```

where n is a sequential number starting at 1, and m is the total number of words generated. This should be followed by a sorted list of pronounceable words, each using every (compound) consonant exactly once. There is no requirement as to how many times each vowel may/must be used. If there are more than 100 words in the output, write just the first 100 words. Words should be written 10 per line for all lines of output (except possibly the last). Leave a blank line between output for each data set.

Sample Input

```
3 a ee i
2 s st
2 a e
2 s t
0
```

Output Corresponding to Sample Input

```
Data set 1:
96 words.
asast asasta asastee asasti aseest aseesta aseestee aseesti asist asista
asistee asisti astas astasa astasee astasi astees asteesa asteesee asteesi
astis astisa astisee astisi eesast eesasta eesastee eesasti eeseest eeseesta
eeseestee eeseesti eesist eesista eesistee eesisti eestas eestasa eestasee eestasi
eestees eesteesa eesteesee eesteesi eestis eestisa eestisee eestisi isast isasta
isastee isasti iseest iseesta iseestee iseesti isist isista isistee isisti
istas istasa istasee istasi istees isteesa isteesee isteesi istis istisa
istisee istisi sast sasta sastee sasti seest seesta seestee seesti
sist sista sistee sisti stas stasa stasee stasi stees steesa
steesee steesi stis stisa stisee stisi

Data set 2:
36 words.
asat asata asate aset aseta asete atas atasa atase ates
atesa atese esat esata esate eset eseta esete etas etasa
etase etes etesa etese sat sata sate set seta sete
tas tasa tase tes tesa tese
```

9. Share and Share Alike

In the spirit of congeniality, players at a round table poker game decide to even out their winnings after a game ends. The scheme involves sharing their winning—each player divides his money into three parts according to a specified pattern *L*:*M*:*R*. *L* indicates the part he shares with his neighbor on the left, *M* indicates the part he keeps for himself, and *R* indicates the part he shares with his neighbor on the right. However, if the left or right portion contains loose change (i.e., partial dollars), the player keeps that, and only gives whole dollar amounts to his/her neighbors.

For example, suppose there are four players seated counterclockwise around the table with winnings of $60, $60, $120, and $90, respectively. If the sharing scheme is 2:3:1, each shares 1/3 (that is, 2/6) of his money with the player on the left, 1/6 of his money with the player on his right, and keeps the rest of his money for himself. After sharing, the money is distributed in the following amounts: $65, $80, $100, and $85. The players may decide to repeat this scheme more than once. If they share a second time, the money is distributed in these amounts: $74, $84, $92, and $80.

Your job is to find how the money is finally distributed among the players after sharing is complete:

Input

There will be one or more games specified. For each game, you will be provided with this input:

- The first line will be the number of players, *N*, where *N* is an integer, $N > 0$
- The next *N* lines will be the original amount held by each player, where each value is a non-negative integer. These are provided in the order the players are seated counterclockwise around the table
- The next line is the sharing scheme in the form *L M R* (three non-negative integers). At least one of *L M R* will be greater than 0.
- The next line is a non-negative integer indicating the number of times the sharing scheme is to be carried out.

A zero for the number of players will indicate end of input, and that game should not be processed.

Output

For each game, output a line with the word "**Game**" and the number of the game, beginning with 1. The next line indicates the distribution of money for the players, with a single space between the numbers. This is to be followed by a blank line.

Sample Input

```
4
60
60
120
90
2 3 1
2
5
100
0
0
0
0
1 5 4
3
0
```

Output Corresponding to Sample Input

```
Game 1
74 84 92 80

Game 2
25 34 25 8 8
```

10. Spell Checker

When you use a spell checker, and the lookup word is wrong, it usually makes suggestions. Our task is, given a list of correctly spelled words (dictionary words) and a possibly misspelled lookup word, which of the dictionary words is/are closest to the lookup word. By "closest" we mean that the lookup word has lowest penalty value relative to the dictionary word. Penalty values are calculated for the following errors:

Penalty	Spelling error	Example lookup word misspellings of the dictionary word HELLO
0	No error	HELLO
2	One extra letter	HELLOT, THELLO, HELLTO
4	Two extra letters	AHELLTO
2	One missing letter	HELO
4	Two missing letters	HLL
1	Adjacent letters interchanged	EHLLO, HLELO
2	One wrong letter	HULLO
4	Two wrong letters	HULUO

Only one error—the one with the lowest penalty—is counted for a lookup word/dictionary word pair.

Input

The input will consist of one or more data sets. Each data set will contain a line containing an integer n ($n < 1000$), followed by n lines each having a dictionary word (consisting of 1 to 100 uppercase letters). This is followed by a line containing an integer m ($1 \leq m \leq 100$), followed by m lines each having a lookup word (consisting of 1 to 100 uppercase letters). The last data set has n=0, and should not be processed.

Output

For each data set, there should be a line like:

`Data set <n>:`

where `<n>` is a sequential data set number starting at 1. This is followed by m lines (one per lookup word) looking like:

`<lookup_word>: <dict_word_1>, <dict_word_2>, …<dict_word_p>.`

The penalty of the lookup word is the lowest possible (and the same) for each of the following list of dictionary words. The list of dictionary words should be in alphabetical order.
If there are no matches, the output should look like:

`<lookup_word>: no suggestions.`

There should be an empty line between data sets.

Sample Input

```
4
CHILL
CHELLO
BELLOW
HELL
5
HELL
CHAMP
CHELL
THELLO
THELLI
2
HI
HO
1
HUM
0
```

Output Corresponding to Sample Input

```
Data set 1:
HELL: HELL.
CHAMP: no suggestions.
CHELL: CHELLO, CHILL, HELL.
THELLO: CHELLO.
THELLI: CHELLO, HELL.

Data set 2:
HUM: no suggestions.
```