0001001110011000111110100100110
0100100010000001101000110100001
1010010101110100010110000000100
COMPUTER AND INFORMATION SCIENCE
1101000011100110110010000100100
0001000110100010001000000010011
1010001101010001010100100100011
0010010001001111000011100010101

1110001001000011001110011010011
0011100001110100000100101010010

Sixteenth Annual
University of Oregon
Eugene Luks Programming Competition

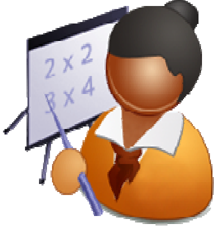
Saturday, April 14, 2012

Problem Contributors

Jim Allen
David Atkins
Gene Luks

Prizes and food provided by Pipeworks

OLD MATH



These days kids use calculators and computers to do most arithmetic, but it is still a good idea to learn and understand how to do arithmetic by hand. For example, to multiply two large numbers by hand, we are taught the technique of calculating the intermediate products formed when you multiply the first number by each digit of the second number, and recording those results with columns aligned to reflect the progressive powers of 10. Then we add up the intermediate results to get the final answer.

```
  123
x 456
  738
 615
492
56088
```

.

For this problem, you are given two numbers that are to be multiplied and you must show the "long multiplication". Use leading blanks so that all digits are aligned properly. Each intermediate product is on a line of its own, except that if a digit in the second operand is zero, then that intermediate product will be combined on the line with the following one. The operands must be followed by a line of dashes, and the final product must be preceded by a line of dashes. If there would be only one intermediate product, then don't show it.

The first line in input specifies the number of multiplications to work out. Each multiplication exercise is on a line and consists of two numbers to be multiplied. Each of the two numbers will be non-negative and less than one billion, and neither number (other than zero itself) will have leading zeroes. Each multiplication solution must be followed by a blank line. The sample output shows the proper output formatting, which you must produce exactly as shown.

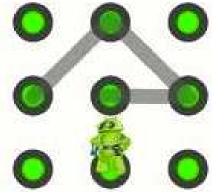
Sample Input

```
3
32 16
124 302
345 6
```

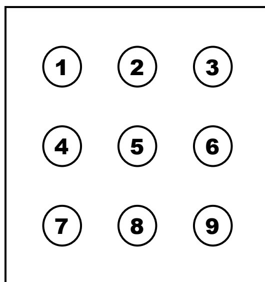
Sample Output

```
 32
 16
---
192
32
---
512

  124
  302
-----
  248
3720
-----
37448

 345
   6
----
2070
```

CONNECT THE DOTS



On an Android phone, you can set a pattern to be traced as the way to unlock the phone. This can be a more intuitive way to unlock the phone than entering a numeric PIN. The pattern is formed by starting with one dot in a 3x3 grid and moving to other dots. Tracing the dots is just another interface for entering a sequence of numbers. At first glance, you might think there is a dot pattern for every possible PIN, but there are some restrictions on tracing the dot pattern. In the sequence described by a dot pattern, no number may occur more than once. And, a pair of dots in the sequence (we'll call them the current and target dots) is subject to this rule: From the current dot, you can move to any other dot that has not been traced yet, provided that any dots on the straight line from the current to the target dot have already been traced. If there is an untraced dot on the straight line from the current dot to the target dot, then it is impossible to get to the target dot without tracing the intermediate dot, so it would be an invalid sequence of numbers.



The dots are numbered, beginning with 1, from left to right and top to bottom, as pictured for a 3x3 grid. The sequence traced by **2-1-5-9** is a valid sequence. However, the sequence **2-1-9-5** is not valid since you can not get from **1** to **9** without first tracing **5**. The sequence traced by **2-1-5-9-5** is also not valid since **5** cannot appear twice in the sequence. On the other hand, the sequence **2-5-1-9** is valid since in this case **5** has already been traced, so it is okay to go through it to get to **9**, which has not yet been traced.

For this problem, you must determine if a sequence of numbers is described by a valid tracing of dots. No number may be repeated in the sequence, and the tracing for the sequence must follow the rules described above. Although Android phones only use a 3x3 array of dots, for this problem you must consider grids of any size.

The first line in input specifies the number of sequence configurations to solve. Each subsequent line describes a configuration and consists of a list of space-separated numbers. The first three numbers M, N, and L, are the number of rows in the grid, the number of columns in the grid, and the length of the sequence, with $1 \le M, N \le 50$ and $1 \le L \le M \times N$. The remaining numbers on the line form the sequence. For each line of input, you must output either "valid" or "invalid" according to whether the sequence follows the rules.

Sample Input

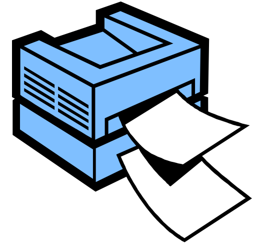
```
6
3 3 4 2 1 5 9
3 3 4 2 1 9 5
3 3 5 2 1 5 9 5
3 3 4 2 5 1 9
4 3 6 1 2 3 10 11 12
4 3 6 1 2 3 7 8 9
```

Sample Output

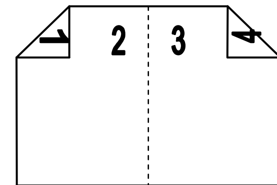
```
valid
invalid
invalid
valid
valid
invalid
```

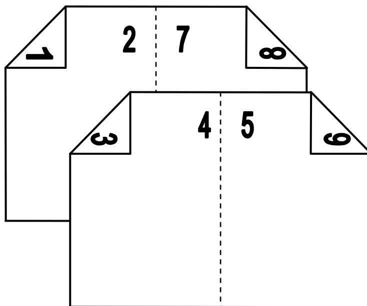
CHAPTER AND PAGE

A conventional duplex printer can be used to print a small book by printing on both sides of the sheets of paper and then folding the sheets in half to form a book. To accomplish this, two pages are printed side by side in landscape mode on each side of a sheet of paper. There will be one fourth as many sheets as there are total pages in the book. Of course, the placement of the pages is very important so that the book is read in the correct order.



For example, suppose the book has four pages. On the single sheet of paper, page 1 would be printed on the back right, page 2 on the front left, page 3 on the front right and page 4 on the back left. This is shown in the drawing to the right. The drawing below shows the placement on two sheets if the book had 8 pages. Note that sheets are output from the printer with the front side down and the first sheet on the bottom so that we can just fold and staple to make the book.





However, there is another complicating factor. The book is formed of chapters, and like many books, we want each chapter to start on an odd numbered page, so that first pages of chapters will always appear on the right side as we turn the pages of the book. This means that sometimes a blank page will have to be printed at the end of one chapter before the start of another. Finally, to keep things simple and tidy, we will add blank pages to the end of the book to ensure each paper sheet is completely filled. All pages (including the blank pages) will have page numbers.

For this problem, you are given the number of pages in each chapter of a book. You are also given a page number. With the printing conditions described above, you are to determine which paper sheet the page will print on, where on the paper sheet it will print, and what will print (i.e., the page of the chapter or a padding blank page). The first line of input specifies the number of puzzles to be solved. Each subsequent line describes a printing puzzle: it consists of a page number P, followed by a number, N, of chapters, and then N numbers representing the number of pages in each chapter. The output should describe for page P what is to be printed on which sheet and should be formatted as shown in the sample.

Sample Input

```
3
2 1 4
4 3 2 1 3
6 3 2 1 3
```

Sample Output

```
Page 2: page 2 of chapter 1 on the front left of sheet 1
Page 4: blank on the front left of sheet 2
Page 6: page 2 of chapter 3 on the back left of sheet 2
```

DROID MATCH

Anakin found R2D2 to be an eager playmate and taught him a game that he had played with his friends on Tatooine. Two youngsters would gather a large pile of sand pebbles and take turns removing either 1 or 2 pebbles. The winner was the one who removed the last pebble. Back on Tatooine, Anakin had been a strong competitor; perhaps his moves were unwittingly influenced by the Force. Nevertheless, Anakin could not dominate R2D2, whose computing speed measured in zettaflops, and he soon tired of the game. On the other hand, R2D2 was hooked and introduced the game to fellow astro droids during R&R (recharging and refitting) periods.

To maintain a challenge, the droids worked with larger and larger random piles of (virtual) pebbles and generalized the rules: for each game a random set M of positive integers was declared so that each move involved removing m pebbles for some $m \in M$. The methodical droids would then play out the full game even though, for such perfect players, the outcome is predetermined from the starting configuration,

For this problem, you are given the initial pile size P and a set M of positive integers including 1 (so that the entire pile would be removable). R2D2 is scheduled to move first. If he is to win the game, you should determine his possible first moves.

The first line of the input will specify the number N of games to be analyzed. Each of the next N lines will indicate the pile size P for a game, with $0 < P < 2^{64}$, followed by the set M of legal moves, with $M \subseteq \{1, 2, 3, \ldots, 21\}$. For each game, the output should be one of the forms

R2D2 will lose.

or

R2D2's first move must be m.

or

R2D2's first move must be one of: $m_1, m_2, \ldots, m_k$.

where $1 \leq m \leq 21$ and $1 \leq m_1 < m_2 < \ldots < m_k \leq 21$. There should be a blank line between games.

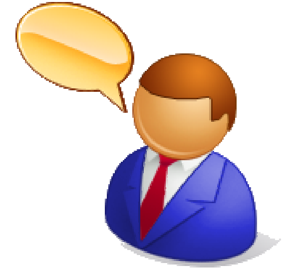
Sample Input

```
6
4 1
5 1 2
9 1 2
12 1 3 6
15 1 4 5 10
12 1 4 5 7 10
```

Sample Output

```
R2D2 will lose.

R2D2's first move must be 2.

R2D2 will lose.

R2D2's first move must be one of: 1, 3.

R2D2's first move must be one of: 1, 4.

R2D2's first move must be one of: 1, 4, 10.
```

JUST SAY NO



You are working on an automated speech recognition system. Fortunately, your application only has to decide what word the customer said from a short list of options. Unfortunately, it is difficult to distinguish certain sounds using waveform analysis. Because of regional variation, vowels are difficult to match, but they are easily recognized and thrown out, and the computer reports the word it has heard as a string of consonant sounds. You must decide which word (from a list) the person most likely said.

To determine this, the probabilities for each consonant sound are given in the table below. For a sound the computer reports, the corresponding row in the table gives the probabilities of what was actually said. For example, the highlighted row in the table gives the probabilities if the computer reports the sound 'd'. The probability the reported 'd' was actually a 'b' is 0.8, and the probability that it was actually a 't' is 0.9.

	p	**b**	**m**	**f**	**v**	**t**	**d**	**s**	**z**	**n**	**S**	**Z**	**k**	**g**	**N**	**ℓ**	**L**	**r**
p	1.0	0.7	0.7	0.5	0.3	0.8	0.6	0.2	0	0	0	0	0.8	0.6	0	0	0	0
b	0.7	1.0	0.9	0.3	0.5	0.6	0.8	0	0.2	0.2	0	0	0.6	0.8	0.2	0	0	0
m	0.7	0.9	1.0	0.3	0.5	0	0.2	0.4	0.7	0.9	0.4	0.5	0	0.2	0.9	0.3	0.3	0.3
f	0.5	0.3	0.3	1.0	0.9	0.2	0	0.7	0.4	0.2	0.7	0.4	0	0	0.2	0	0	0
v	0.3	0.5	0.5	0.9	1.0	0	0.2	0.4	0.7	0.3	0.4	0.7	0	0	0.3	0.1	0.1	0.1
t	0.8	0.6	0	0.2	0	1.0	0.9	0.5	0.3	0.3	0.3	0.1	0.8	0.6	0.1	0	0	0
d	0.6	0.8	0.2	0	0.2	0.9	1.0	0.3	0.5	0.4	0.1	0.3	0.6	0.8	0.3	0	0	0
s	0.2	0	0.4	0.7	0.4	0.5	0.3	1.0	0.9	0.3	0.9	0.6	0.2	0	0.4	0.1	0.1	0.1
z	0	0.2	0.7	0.4	0.7	0.3	0.5	0.9	1.0	0.5	0.6	0.9	0	0.2	0.7	0.2	0.2	0.2
n	0	0.2	0.9	0.2	0.3	0.3	0.4	0.3	0.5	1.0	0.1	0.3	0	0.1	0.9	0.3	0.3	0.3
S	0	0	0.4	0.7	0.4	0.3	0.1	0.9	0.6	0.1	1.0	0.9	0.2	0	0.4	0.2	0.2	0.2
Z	0	0	0.5	0.4	0.7	0.1	0.3	0.6	0.9	0.3	0.9	1.0	0	0.2	0.6	0.3	0.3	0.3
k	0.8	0.6	0	0	0	0.8	0.6	0.2	0	0	0.2	0	1.0	0.9	0.3	0	0	0
g	0.6	0.8	0.2	0	0	0.6	0.8	0	0.2	0.1	0	0.2	0.9	1.0	0.5	0	0	0
N	0	0.2	0.9	0.2	0.3	0.1	0.3	0.4	0.7	0.9	0.4	0.6	0.3	0.5	1.0	0.3	0.3	0.3
ℓ	0	0	0.3	0	0.1	0	0	0.1	0.2	0.3	0.2	0.3	0	0	0.3	1.0	0.8	0.7
L	0	0	0.3	0	0.1	0	0	0.1	0.2	0.3	0.2	0.3	0	0	0.3	0.8	1.0	0.6
r	0	0	0.3	0	0.1	0	0	0.1	0.2	0.3	0.2	0.3	0	0	0.3	0.7	0.6	1.0

Sometimes the computer will interpret background noise as additional sounds so there could be extra consonants in the string as well as the wrong consonants. For each utterance, you must compute the best correspondence between the word the computer heard and the possible choices. The best correspondence is determined by the highest possible sum of the chances of getting it right. For each sound in the utterance it might be background noise, in which case we remove it and add nothing, or it has a probability of matching a consonant according to the table. For example, the best correspondence between the computer hearing `pmpt` and the choice `dpt` is found by throwing out the `m` and computing `p`→`d`, `p`→`p` and `t`→`t` for a total of 0.6+1.0+1.0=2.6. If instead we throw out the initial `p`, then we get `m`→`d`, `p`→`p` and `t`→`t`, for a correspondence of 0.2+1.0+1.0=2.2, which is not as good, so we use 2.6. If the computer heard `pmpt` there is no correspondence with the choice `pmplt`.

The input will start with one number on a line indicating the number of recognition decisions to solve. Each recognition decision consists of a line of a list of space separated "words". The

first word is the utterance the computer heard, and the remaining words on the line are the allowed choices. For each recognition decision, your program should output which choice has the highest correspondence with what the computer heard. Utterances and choices will consist of only the letters from the table (vowels and semi-vowels will already have been removed). If there is no best choice (i.e., the best correspondence score is zero), then your program should output “What?”.

The table is in the file /home/users/contest/2012/sounds.dat, and your program may open that file to read in the table when it runs.

Sample Input

```
6
ptmLt mlt pz rs
pz mlt ptmLt rs
pdmlt mlt ptmLt rs
rgt lft rt strt
tm dn rm tn
pmpd prmpt pmplt
```

Sample Output

```
mlt
rs
ptmLt
rt
tn
What?
```

For these samples, the choices are

1. omelet, peas, or rice; the computer heard “patty melt” and matched it with omelet.
2. omelet, patty melt, or rice; the computer heard “peas” and matched it with rice.
3. omelet, patty melt, or rice; the computer heard “paddy melt” and matched it with patty melt.
4. left, right, or straight; the computer heard “right”, with loud background noise and matched it with right.
5. din, rhyme, or tin; the computer heard “tim” and matched it with tin.
6. prompt or pamphlet; the computer heard “pumped” and there is no correspondence