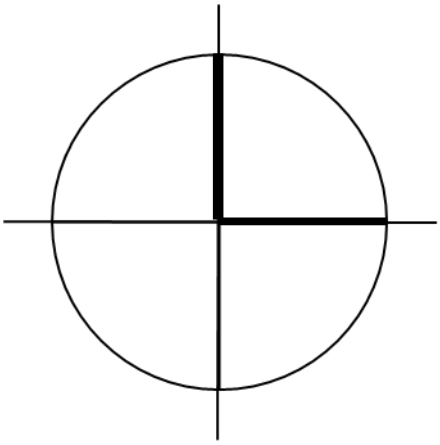
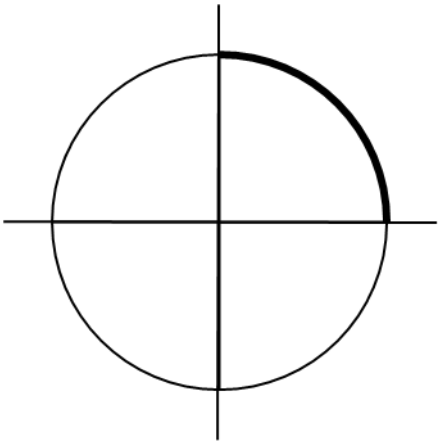
Problem A: Driving Around the Beltway.

Delaware, as you may have noticed, is north of Columbus. When driving from Delaware to the Muskingum Contest in the fall, we drove down to Columbus on I-71 (which runs north-south), then got on I-70 going east. Surrounding Columbus is I-270, the Columbus Beltway. If we assume that the beltway is a circle, and that there are two roads bisecting the beltway, we have two options.
Option 1: Drive to the middle, then switch to the other road (basically driving the radius of the beltway twice, once into the middle, and once out.)



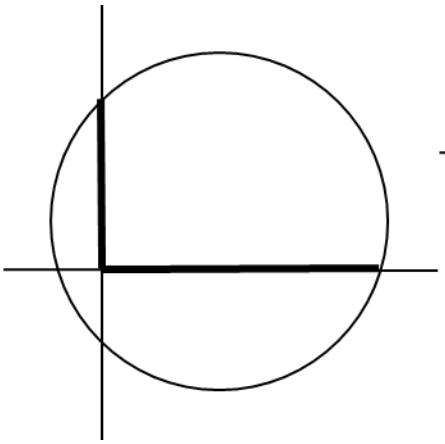
Option 2: Drive around the beltway, and pick up I-70 when it exits the beltway (this method makes us drive ¼ of the circumference of the beltway.)



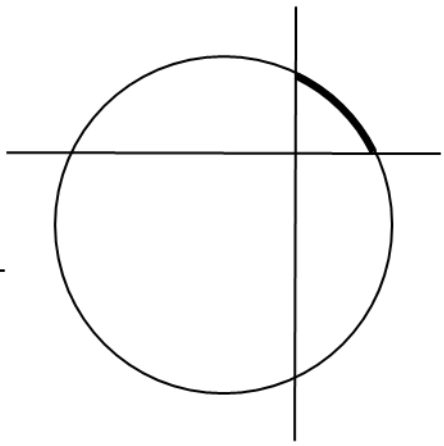
If the radius of the beltway is r miles, we'd expect to drive 2*r miles with Option 1, and ½*π*r miles (≈1.57 r miles) with Option 2. Thus, we'd expect Option 2 to be the better choice.

Mapquest, however, chose Option 1 for us. The reason for this is that the intersection between the north-south and east-west roads isn't in the center of the circle. (It's also the case that I-270 isn't exactly circular, but we'll ignore that for this problem.)

It turns out that the question of which option is better depends on where exactly both the vertical and horizontal roads intersect. Your task is to write a program that determines the best path for a variety of potential situations.



(Going through the middle is shorter)



(Going around the beltway is shorter)

For the purposes of this problem, imagine that the beltway is a circle of radius 100 centered around he origin (0,0). The north-south road will be a line of the equation x=*p*, where *p* is a number between -100 and 100. The east-west road will be a line of the equation y=*q*, where *q* is a number between -100 and 100. Note that the north-south road will always be perpendicular to the east-west road.

Input:
There will be several input instances. Each input will contain two integers, containing values for *p* and *q*.
We will always be traveling from north of the beltway to east of the beltway.
Values of *p* and *q* that are outside of the range -100 to 100 will signify the end of the input.

Output:
For each test case *i*, starting at 1, output either:
Case *i*: Go around the beltway.
or
Case *i*: Go through the middle.

Sample Input:
0 0
42 42
-85 28
200 200

Sample Output:
Case 1: Go around the beltway.
Case 2: Go around the beltway.
Case 3: Go through the middle.

Problem B: That's Not My $Object

There's a popular series of children's books with titles like *That's Not My Dinosaur* . (A quick search of Amazon found at least 20: *That's Not My Puppy, That's Not My Truck, That's Not My Princess*, and my personal favorite, *That's Not My Donkey.)*

All of the books are the same, and have the same general format. There are several pages of "That's not my (whatever)… its (something) is too (property). " Eventually, the book ends with: "THAT'S my (whatever)! It's (something) is so (property)."

So, for example, the dinosaur book goes as follows:
"That's not my dinosaur… Its body is too squashy.
That's not my dinosaur… Its tail is too fuzzy.
…
THAT'S my dinosaur! Its spines are so soft."

This formula is so easy, that you can generate a book with a simple program. So get to it!

Input:
The input will begin with an integer, n, denoting the number of books to write.
Each of the n books will have input in the following format:

- The first line will consist of a string, s (the object the book is about), and an integer p, for the number of pages.
- There will then follow p lines of the form:
  Feature/Adjective
  Denoting the details of each page. "Feature" and "Adjective" will both be strings without spaces, separated by the '/' character.

So the format of each page (except the last) will be:
"That's not my *s*.. Its *Feature* is too *Adjective*".
(where s, Feature, and problem are all replaced with the individual values given by the input"
The last of the p lines will be for the last page, which is of the format:
"THAT'S my *s*.. Its *Feature* is so *Adjective*"

Output:
For each book, output the line:
Book i:
(where I is the current book number, starting from 1)
Then the pages of each book.
Leave a blank line after each book (<u>including the last one</u>)

Sample Input:

2
dinosaur 6
body/squishy
tail/fuzzy
teeth/bumpy
flippers/slippery
horns/rough
spines/soft

puppy 6
coat/hairy
tail/fluffy
paws/bumpy
collar/shiny
ears/shaggy
nose/squashy

Sample Output :

(note the use of “is” when grammatically it should be “are”. That’s what you should do)
Book 1:
That's not my dinosaur... Its body is too squishy.
That's not my dinosaur... Its tail is too fuzzy.
That's not my dinosaur... Its teeth is too bumpy.
That's not my dinosaur... Its flippers is too slippery.
That's not my dinosaur... Its horns is too rough.
THAT’S my dinosaur! Its spines is so soft.

Book 2:
That's not my puppy... Its coat is too hairy.
That's not my puppy... Its tail is too fluffy.
That's not my puppy... Its paws is too bumpy.
That's not my puppy... Its collar is too shiny.
That's not my puppy... Its ears is too shaggy.
THAT’S my puppy! Its nose is so squashy.

Problem C: Curling Scoring

Curling is an Olympic sport that is played in a series of "ends" (sort of like "innings" in baseball). In each end, at most one team can score (and it's possible that neither team can score).

At many local clubs, the scoreboard of a curling match is constructed as a table, where the rows are the score, and the values in the table are the ends in which the score occurred.

Here is an example score table for a curling match

	1	2	3	4	5	6	7	8
Team A	0	0	1	0	3	4	0	0
Team B	2	0	0	6	0	0	0	0

The score is across the top, and the end in which the score happened is inside the table.
So, in this match:
Team A scored 3 in the first end (because of the 1 in column 3 for team A)
Team B scored 1 in the second end (currently, Team A is winning 3-1)
Team A scored 2 more in the third end (making the score 5-1- notice how the total score is 5, and that's the column we see a 3)
Team A scored again in the fourth end (making the score 6-1)
Nobody scored in the fifth end
Team B scored 3 in the sixth end (making the final score 6-4, with team A winning)

Notice how the rightmost non-zero number is the final score for both teams. Also notice that Team A won because they have a number in the higher column (column 6)

Write a program that, given a scoreboard in the above format, describes all of the scoring for each end of a curling match.

Input:
There will be several input instances, each one beginning with an integer c, indicating the number of columns in the scoring table. A value of c=0 denotes the end of input.

Each input instance will consist of two rows of c integers, each separated by spaces. The first row holds the information of Team A, the second row holds the information for team B.

The entries will comprise a legal curling match (at most one team scores in each end, all scores will be non-negative) and each match will have at most 10 ends (and perhaps less)

Output:
For each match, output the line:
Match i:
(starting from 1)
For each end in the match, output the scoring in the end, as well as the current score. If a team scores, output:
End j: Team A/B scores k, score is now A-B
If there is no score, output:
End j: No scoring, score is now A-B.

When outputting the current score of the match, always put team A's score first, even if they are losing.

Put a blank line after each match, including the last one

Sample Input:
8
0 0 1 0 3 4 0 0
2 0 0 6 0 0 0 0
7
1 0 2 0 0 3 0
0 4 5 0 6 0 7
0

Sample Output:
Match 1:
End 1: Team A scores 3, score is now 3-0.
End 2: Team B scores 1, score is now 3-1.
End 3: Team A scores 2, score is now 5-1.
End 4: Team A scores 1, score is now 6-1.
End 5: No scoring, score is now 6-1.
End 6: Team B scores 3, score is now 6-4.

Match 2:
End 1: Team A scores 1, score is now 1-0.
End 2: Team A scores 2, score is now 3-0.
End 3: Team A scores 3, score is now 6-0.
End 4: Team B scores 2, score is now 6-2.
End 5: Team B scores 1, score is now 6-3.
End 6: Team B scores 2, score is now 6-5.
End 7: Team B scores 2, score is now 6-7.

Problem D: Friend Finding

(Shamelessly stolen from Michelle Craig at the University of Toronto)

If you've ever looked for books, movies, or something similar online, you've seen many sites give you the ability to rate items. The ratings are then compared against other people who make similar ratings. Then, new items are suggested for you under the assumption that if someone who liked the things that you like rated this new item highly, you're likely to like the same new item.

A possible extension of this idea is to suggest "friends" based on recommendations. A good choice for a potential friend is someone who likes most of the same things that you do.

Imagine that we have a pool of N items $\{x_1,x_2,\ldots x_n\}$, each with a rating $\{r_1,r_2,\ldots r_n\}$. Each rating is a letter:
'h'ated it
'd' isliked it
'n'eutral
'l'iked it
'w'onderful
'u'nseen
We can give each rating a numerical score (we can call this n_i):
h=-5, d=-3,n=1, l=3,w=5, u=0

Now, we can get a similarity score for each item by multiplying the n_i of each person together, and then get a similarity score for a person by adding these products together. (If you've taken Linear Algebra, this is the "dot product"). We can then use this similarity score to rate potential friends.

Input:
There will be multiple problem instances. Each instance starts with two numbers: N for the number of books, and M for the number of raters. Values of M =N = 0 signify the end of input.
Each input instance will then have M+1 lines, one for each rater, and the last one for the user. Each of these lines will start with a string (the name of the rater), and then N rating characters, in the range described above. The last of the M+1 lines will be the ratings of the user, to be compared against the M previous raters for similarity.

Output:
For each input instance i, starting with 1, output the line:
Case i: Potential friends for s
M_1 S_1
M_2 S_2
…
M_m S_n

"S" is the name of the user, and each of the M_i are names of the raters, listed in decreasing order of similarity. If two people have the same similarity score, output the tied people in any order. The S_i is the similarity score for user i.

Leave a blank line after each case including the last one

Sample input:

3 3
Suelyn n w d
Bob w d w
Kalid n l u
Rabia w l h

5 4
Alice w l n d h
Bob h d n l w
Carol u u l u n
Dave l u w u d
Evelyn u u u l h
0 0

Sample Output

Case 1: potential friends for Rabia
Suelyn 35
Kalid 14
Bob -9

Case 2: potential friends for Evelyn
Alice 16
Dave 15
Carol -5
Bob -16

Problem E: An Exercise in Rationalization

Being a fan of a sports team is often a frustrating endeavor. After all, most of the time, some other team wins the championship, and you have to watch some other person celebrating. Luckily, in many sports, the team that won the championship lost a game to someone along the way. And that someone probably lost to someone else. If you can find a chain of these teams that leads back to your team, you can claim "Well, my team beat team A, which beat team B, (insert possibly long chain of games), .. which beat the champion! Obviously my team is better than those so-called 'champions'!"

It helps when making these claims to have as short a sequence of games as possible. It's hard to convince someone that your team is really the best if you have to make a chain of 20 games to prove it.

Input:
There will be multiple input instances, and each instance will have multiple teams to compare.
Each input instance will begin with a number N, for the number of teams in the league. A value of N=0 denotes the end of input.

There will be N lines of integers, one for each team. Each line will start with a non-negative integer w, denoting the number of teams to follow. Then there will be w integers, denoting the teams that were beaten. The first line will be teams beaten by team 1, the second line will be teams beaten by team 2, and so on. Each team will be represented as a number between 1 and N.

After the N lines, there will be an integer k, and k pairs of integers. For each pair of integers a b, you are trying to find a chain of games where team a is "better" than team b.

Output:
For each league *i* (starting from 1), output the line:
League *i*:
For each pair in the league output:
Team a is better than team b in k games : a->g_1->g2-> .. g_{k-1}-> b
(the '->' is a dash followed by a greater-than sign with no spaces)
Where the g_i are the teams in the shortest chain. If there is a tie for the shortest game, choose the smallest numbered team for g_1, then the smallest team for g_2, and so on.

Put a blank line after each league <u>including the last</u>

Sample Input

```
4
1 2
1 3
1 4
1 1
5
1 3
1 4
1 2
3 1
3 2

5
4 2 3 4 5
3 3 4 5
2 4 5
1 5
0
3
1 5
5 4
4 2
0
```

Sample Output:

```
League 1:
Team 1 is better than team 3 in 2 games: 1->2->3
Team 1 is better than team 4 in 3 games: 1->2->3->4
Team 1 is better than team 2 in 1 games: 1->2
Team 3 is better than team 1 in 2 games: 3->4->1
Team 3 is better than team 2 in 3 games: 3->4->1->2

League 2:
Team 1 is better than team 5 in 1 games: 1->5
Team 5 is not better than team 4.
Team 4 is not better than team 2.
```

Problem F: Finals Scheduling

Many colleges and universities schedule their final exams by time slot. (So, all 10:00 MWF classes take their finals at the same time, all 11:00 MWF classes will have their finals at some other time, and so on). Because of the limited time to give finals, multiple slots will have their finals in the same day (For the purposes of this problem, there will be at most two finals on each day). Some students will be unlucky enough to have two tests in the same day, and we would like to generate a schedule that reduces the number of these students.

Input:
There will be multiple input instances. Each instance will begin with an integer n (n <=8) that denotes the number of time slots to schedule. A value of n=0 signifies the end of input. If n > 0, there will then be an integer m (m <= 16), denoting the number of students, and m lines will follow. Each of the m lines will contain an integer c, for the number of classes the student is taking, and then c integers between 1 and n. These integers represent the time slots in which the student is taking courses.

Output:
For each input case I, output the line:
Case i: Found a schedule with k annoyed students.
..where k is the smallest possible number of students having a day with multiple finals in a schedule with n/2 days of finals. (If n is odd, then there will be one day that only has one test).
Always use "students" even if the word "student" would be grammatically correct.

Sample Input:
```
4 5
3 1 2 3
4 1 2 3 4
2 1 3
3 2 3 4
2 2 4
6 4
3 1 2 3
3 1 3 5
4 1 2 3 4
3 1 5 6
0
```

Sample Output:
Case 1: Found a schedule with 3 annoyed students.
Case 2: Found a schedule with 1 annoyed students.