## Problem A – "To and Fro"

(Problem appeared in the 2004/2005 Regional Competition in North America – East Central.)

Mo and Larry have devised a way of encrypting messages. They first decide secretly on the number of columns and write the message (letters only) down the columns, padding with extra random letters so as to make a rectangular array of letters. For example, if the message is "There's no place like home on a snowy night" and there are five columns, Mo would write down

```
t o i o y
h p k n n
e l e a i
r a h s g
e c o n h
s e m o t
n l e w x
```

Note that Mo includes only letters and writes them all in lower case. In this example, Mo used the character 'x' to pad the message out to make a rectangle, although he could have used any letter. Mo then sends the message to Larry by writing the letters in each row, alternating left-to-right and right-to-left. So, the above would be encrypted as

```
toioynnkpheleaigshareconhtomesnlewx
```

Your job is to recover for Larry the original message (along with any extra padding letters) from the encrypted one.

## Input

There will be multiple input sets. Input for each set will consist of two lines. The first line will contain an integer in the range 2...20 indicating the number of columns used. The next line is a string of up to 200 lower case letters. The last input set is followed by a line containing a single 0, indicating end of input.

## Output

Each input set should generate one line of output, giving the original plaintext message, with no spaces.

## Sample Input

```
5
toioynnkpheleaigshareconhtomesnlewx
3
ttyohhieneesiaabss
0
```

## Sample Output

```
theresnoplacelikehomeonasnowynightx
thisistheeasyoneab
```

## Problem B – "Lenny's Lucky Lotto Lists"

(This problem appeared in the 2004/2005 Regional Competition in North America – Greater New York.)

Lotto is a lottery, typically with an accumulating jackpot, in which participants play numbers of their choice in a random drawing. Lenny likes to play the lotto in Lincoln county Louisiana. In the game, he picks a list of *n* numbers in the range from 1 to *m*. If his list matches the drawn list, he wins the big prize, a lifetime supply of large lemons.

Lenny has a scheme that he thinks is likely to be lucky. He likes to choose his list so that each number in it is at least twice as large as the one before it. So, for example, if $n = 4$ and $m = 10$, then the possible lucky lists Lenny could like are:

```
1 2 4 8
1 2 4 9
1 2 4 10
1 2 5 10
```

Thus Lenny has 4 lists to choose from.

Your job, given *n* and *m*, is to count how many lucky lists Lenny has to choose from.

## Input

The first line of input is a single non-negative integer, which is the number of data sets to follow. All data sets should be handled identically. The next lines, one per data set, contain two integers, *n* and *m*. You are guaranteed that $1<=n<=10$ and $1<=m<=2000$ and $n<=m$.

## Output

For each data set, print a line like the following:

`Data set` *`i`*`:` *`n m number`*

where *i* is the data set number (beginning with 1), and *number* is the maximum number of lucky lists corresponding to the provided values of *n* and *m*.

## Sample Input

```
1
4 10
```

## Sample Output

```
Data set 1: 4 10 4
```

## Problem C – "A DP Problem"

(This problem was originally taken from the 2003/2004 Regional Competition in Asia – Tehran; the problem was modified by David Bremner and Mike Fleming.)

In this problem, you are to solve a very easy linear equation with only one variable *x* with no parentheses! An example of such an equation is the following:

$$2x - 16 + 5x + 380 = 98x$$

An expression, in its general form, will contain a '=' character with two expressions on its sides. Each expression is made up of one or more terms combined by '+' or '-' operators. No unary plus or minus operators are allowed in the expressions. Each term is either a single integer, or an integer followed by the lower-case character x or the single character *x* which is equivalent to 1*x*. The terms and operators in the equations are separated by blank spaces.

You are to write a program to find the value of *x* that satisfies the equation. Note that it is possible for the equation to have no solution or have infinitely many. Your program must detect these cases too.

## Input:

The number on the first input line, *t* ($1 <= t <= 10$), is the number of test cases, followed by *t* lines of length at most 255 each containing an equation. There is a blank character between each term and operator in the equations and the variable is always represented by the lower-case character 'x'. The coefficients are integers in the range (0..1000) inclusive.

## Output:

The output contains one line per test case containing the solution of the equation. If *s* is the solution to the equation, the output line should contain *s* if *s* is an integer or the string 'NOTINTEGER' if *s* is not an integer. The output should be 'IMPOSSIBLE' or 'IDENTITY' if the equation has no solution or has infinite solutions, respectively. Note that the output is case-sensitive.

## Sample Input:

```
3
2x - 16 + 5x + 380 = 98x
3x - 2 = 0
x + 2 = 2 + x
```

## Sample Output:

```
4
NOTINTEGER
IDENTITY
```

## Problem D – "Two Ends"

(This problem appeared in the 2005/2006 Regional Competition in North America – East Central.)

In the two-player game "Two Ends", an even number of cards is laid out in a row. On each card, face up, is written a positive integer. Players take turns removing a card from either end of the row and placing the card in their pile. The player whose cards add up to the highest number wins the game. Now one strategy is to simply pick the card at the end that is the largest - we'll call this the greedy strategy. However, this is not always optimal, as the following example shows: (The first player would win if she would first pick the 3 instead of the 4.)

3 2 10 4

You are to determine exactly how bad the greedy strategy is for different games when the second player uses it but the first player is free to use any strategy she wishes.

## Input

There will be multiple test cases. Each test case will be contained on one line. Each line will start with an even integer *n* followed by *n* positive integers. A value of *n* = 0 indicates end of input. You may assume that *n* is no more than 1000. Furthermore, you may assume that the sum of the numbers in the list does not exceed 1,000,000.

## Output

For each test case you should print one line of output of the form:

`In game` *m*`, the greedy strategy might lose by as many as` *p* `points.`

where *m* is the number of the game (starting at game 1) and *p* is the maximum possible difference between the first player's score and second player's score when the second player uses the greedy strategy. When employing the greedy strategy, always take the larger end. If there is a tie, remove the left end.

## Sample Input

```
4 3 2 10 4
8 1 2 3 4 5 6 7 8
8 2 2 1 5 3 8 7 3
0
```

## Sample Output

```
In game 1, the greedy strategy might lose by as many as 7 points.
In game 2, the greedy strategy might lose by as many as 4 points.
In game 3, the greedy strategy might lose by as many as 5 points.
```

## Problem E – "Connect"

(This problem appeared in the 2005/2006 Regional Competition in North America – Pacific Northwest.)

Your task is to decide if a specified sequence of moves in the board game Connect ends with a winning move.

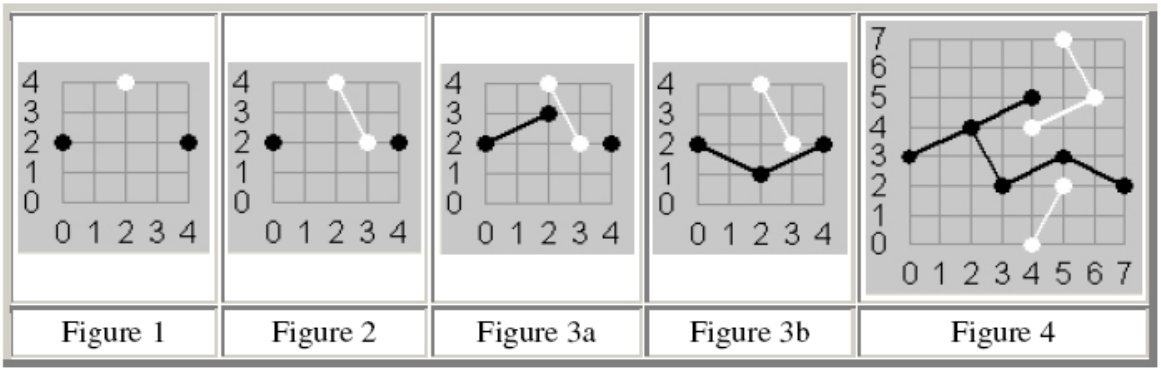


Figure 1 | Figure 2 | Figure 3a | Figure 3b | Figure 4

In this version of the game, different board sizes may be specified. Pegs are placed on a board at integer coordinates in the range [0, *N*]. Players Black and White use pegs of their own color. Black always starts and then alternates with White, placing a peg at one unoccupied position (*x*, *y*). Black's endzones are where *x* equals 0 or *N*, and White's endzones are where *y* equals 0 or *N*. Neither player may place a peg in the other player's endzones. After each play, the latest position is connected by a segment to every position with a peg of the same color that is a chess knight's move away (2 away in one coordinate and 1 away in the other), provided that a new segment will touch no segment already added, except at an endpoint. Play stops after a winning move, which is when a player's segments complete a connected path between the player's endzones.

For example, Figure 1 shows a board with $N = 4$ after the moves (0,2), (2,4), and (4,2). Figure 2 adds the next move (3,2). Figure 3a shows a poor next move of Black to (2,3). Figure 3b shows an alternate move for Black to (2,1) which would win the game.

Figure 4 shows the board with $N = 7$ after Black wins in 11 moves:

(0, 3), (6, 5), (3, 2), (5, 7), (7, 2), (4, 4), (5, 3), (5, 2), (4, 5), (4, 0), (2, 4)

## Input

The input contains from 1 to 20 datasets followed by a line containing only two zeroes, '`0 0`'. The first line of each dataset contains the maximum coordinate $N$ ($3 < N < 21$) and the number of total moves, $M$ ($4 < M < 250$), with $M$ odd, so Black will always be the last player. The dataset ends with one or more lines each containing two or more coordinate pairs, with a total of $M$ coordinate pairs. All numbers on any line will be separated by blanks. All data will be legal. There will never be a winning move before the last move.

## Output

The output contains one line for each data set: '`yes`' if the last move is a winning move and '`no`' otherwise.

## Sample Input

```
4 5
0 2 2 4 4 2 3 2 2 3
4 5
0 2 2 4 4 2 3 2 2 1
7 11
0 3 6 5 3 2 5 7 7 2 4 4
5 3 5 2 4 5 4 0 2 4
0 0
```

## Sample Output

```
no
yes
yes
```

## Problem F – "Auctions R Us"

(This problem appeared in the 2004/2005 Regional Competition in North America – Mid Atlantic.)

Having run into trouble with current online auctions and buyers that win auctions and then back out, you decide to open a new enterprise that has the bidders deposit funds before they may bid on any item. If they win an auction, the amount they bid is immediately (that second!) deducted from their account. (The problem of sellers that don't deliver the items will be left for another day.)

You must write a program to implement the rules of this auction. You will be auctioning off a number of items, each of which will have a reserve price that must be met. Each of your bidders will deposit funds with you, and you must match these funds with items they successfully bid for. You will write a program that tracks the auctions during a single day and outputs the results of each auction.

**Auction Rules**

You are guaranteed:

- No two items will have the same end time.
- No two bids will have the same bid time.
- No price, bid, or account balance will be negative.

Bidder numbers and item numbers are unique within each category, but a bidder may have the same number as an item. Bidder and item numbers are not necessarily assigned sequentially.

An auction is won by the highest bid that:

- arrives no later than the second the auction ends.
- is greater than or equal to the minimum price for the item
- has at least the bid amount remaining in the bidder's account at the instant the auction ends.

## Input

There are 3 sections in each dataset, describing the items available for bid, the registered bidders, and the bids made during the auction.

**Items**

- A single line containing the number of items, *i*
- *i* lines, one for each item of the form:

  < *item number* > < *minimum price* > < *auction end time* >

  where *item number* is a non-negative integer, *minimum price* is specified to the penny (0.01), and < *auction end time* > is in 24 hour format of the form `*XX*:*YY*:*ZZ*' where *XX* is in hours from 00 to 23, *YY* is in minutes from 00 to 59, and *ZZ* is in seconds from 00 to 59.

**Bidders**

- A single line with the number of bidders registered, *j*
- *j* lines of bidder data of the form:

  < *bidder number* > < *account balance* >

  where *bidder number* is a non-negative integer and *account balance* is specified to the penny (0.01).

**Bids**

- A single line with the number of bids received, *k*
- *k* lines of bid data of the form:

  < *item # being bid on* > < *bidder number* > < *bid amount* > < *bid time* >

  where all fields are formatted as described above.

Input will be terminated with a dataset with 0 items. This test case shouldn't be processed.

## Output

Output one line for each item being auctioned, in order of their auction finish time, listing

`Item <`*item number*`> Bidder <`*bidder number*`> Price <`*winning bid*`>`

If there is not a winning bid for an item, for that item output

`Item <`*item number*`> Reserve not met.`

Print a blank line between datasets.

## Sample Input

```
2
1 5.00 05:06:27
2 25.00 15:30:11
2
11 37.37
22 55.55
3
1 11 60.00 04:03:01
2 11 26.00 00:18:03
2 22 27.00 09:03:05
0
```

## Sample Output

```
Item 1 Reserve not met.
Item 2 Bidder 22 Price 27.00
```

## Problem G – "Triangle Cuts"

(This problem appeared in the 2004/2005 Regional Competition in North America – Mid Central; the problem was modified slightly by Mike Fleming and Natalie Webber.)

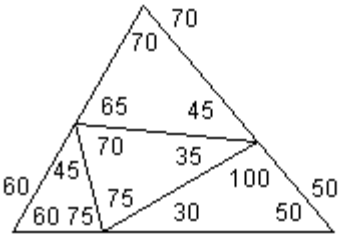

Figure 1

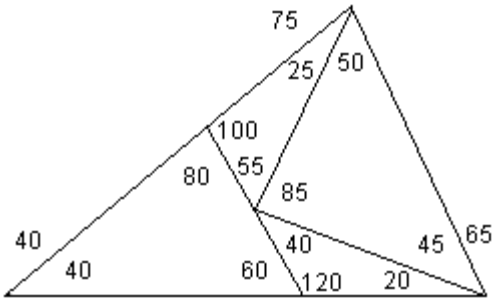

Figure 2

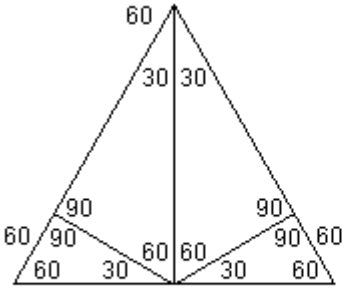

Figure 3

**Disallowed Pattern**

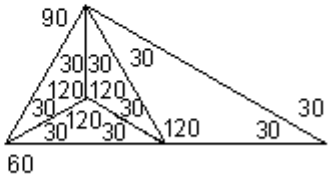

Figure 4

A computer science professor was watching his young daughter use scissors to cut large triangular pieces of paper that were green on one side and white on the other. She always kept the white side of the paper up, so the green side was never visible, and she always followed the same general procedure. Starting with one large white triangle, she would always make exactly three straight cuts, each of which separated one piece of paper into two, and end up with exactly four smaller white triangles. Then she would put the small triangles aside and start over with a new large triangle.

The first three figures show some of the possibilities. All angles are in degrees. In Figures 1 and 2, no cut goes all the way from one vertex of the original large triangle to the opposite side. In fact, these two figures (including any rotations thereof) show the ***only*** ways that she could produce four triangles without making such a cut. The pattern in Figure 4 can't happen, because her cuts always separate the current piece of paper into two pieces; after she cuts off the rightmost triangle in Figure 4, none of the remaining cuts will separate the remaining triangle into two pieces.

The professor imagined a room full of children producing piles of triangles using different methods, and wondered if he could identify his daughter's work. Given the exact size and shape of four small triangles, could he tell if she could have produced them from one large triangle? After a moment's thought he realized that he could simplify the problem and consider only the shapes of the triangles, noting that if the shapes were compatible, there would always be some appropriate sizes for the small triangles. This led to the following problem for you to solve: given the angles of a large triangle and four smaller ones, is it possible that the small triangles could have been obtained from the large one using the procedure his daughter was following?

## Input

The input contains from 1 to 30 datasets followed by a line containing only "0 0 180". Each dataset contains 15 positive integers separated by single blanks on one line. The first 3 integers within each dataset represent the large triangle and the remaining 12 integers represent the four smaller triangles. Each integer is less than 180. Each group of three integers represents the vertex angles for one triangle, expressed in degrees. Assume that none of the triangles is flipped over so the green side of the paper is up, and the vertex angles are listed in clockwise order around each individual triangle.

The first four datasets of the sample input below correspond to the figures above.

## Output

The output contains one line for each data set. If the child's cutting play, as described above, started with a triangle with the first three angles, and could have ended up with triangles matching the last four triangles in the dataset, then the line contains "yes". Otherwise the line contains "no".

## Sample Input

```
60 70 50 30 100 50 75 70 35 75 60 45 45 65 70
40 75 65 60 40 80 20 120 40 45 85 50 25 55 100
60 60 60 30 60 90 30 60 90 90 60 30 90 60 30
30 60 90 30 120 30 30 120 30 30 120 30 30 120 30
60 70 50 30 100 50 75 70 35 75 60 45 70 65 45
0 0 180
```

## Sample Output

```
yes
yes
yes
no
no
```