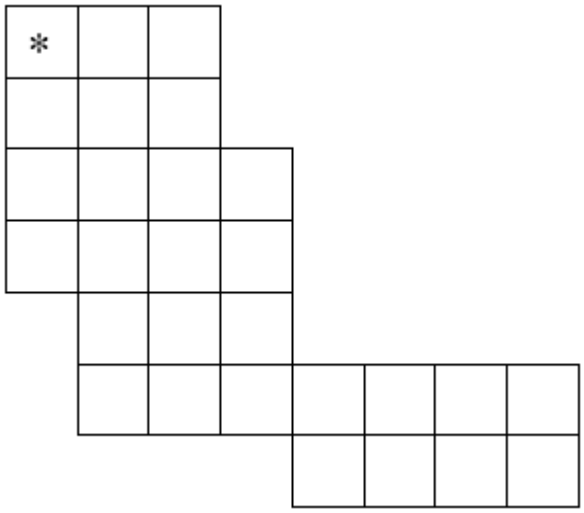
DKC^3 2011 LONG COMPUTING PROBLEMS

1. Good Knight (75 pts)

A knight is a piece used in the game of chess. The chessboard itself is square array of cells. Each time a knight moves, its resulting position is two rows and one column, or two columns and one row away from its starting position. Thus a knight starting on row r, column c – which we'll denote as (r,c) – can move to any of the squares (r-2,c-1), (r-2,c+1), (r-1,c-2), (r-1,c+2), (r+1,c-2), (r+1,c+2), (r+2,c-1), or (r+2,c+1). Of course, the knight may not move to any square that is not on the board.

Suppose the chessboard is not square, but instead has rows with variable numbers of columns, and with each row offset zero or more columns to the right of the row above it. The figure below illustrates one possible configuration. How many of the squares in such a modified chessboard can a knight, starting in the upper left square (marked with an asterisk), not reach in any number of moves without resting in any square more than once?



If necessary, the knight is permitted to pass over regions that are outside the borders of the modified chessboard, but as usual, it can only move to squares that are within the borders of the board.

Input

There will be 10 cases to consider. The input for each case begins with an integer n, between 1 and 10, that specifies the number of rows in the modified chessboard. Following n there will be n pairs of integers, with the ith pair corresponding to the ith row of the chessboard. The first integer of each pair indicates the number of squares skipped at the beginning of the row. The second integer indicates the number of squares in the row (which will always be at least 1).

For example, input for the case illustrated by the chessboard shown above would be:

7 0 3 0 3 0 4 0 4 1 3 1 7 4 4

The maximum dimensions of the board will be 10 rows and 10 columns. That is, any modified chessboard specified by the input will fit completely on a 10 row, 10 column board.

Output

For each input case, display the number of squares that the knight cannot reach.

There will be 10 test cases, each separated by an asterisk.

Input File: C:\DKC3\KnightIn.txt
Output File: C:\DKC3\KnightOut.txt

Input

```
7 0 3 0 3 0 4 0 4 1 3 1 7 4 4
3 0 3 0 3 0 3
```

Output

```
4
1
```

2. Epic Booty^2 (75 pts)

You are on an epic quest to not only save the Princess, but also make yourself filthy stinking rich. Somewhere in a far-off tower lies your true love (booty^1) as well as several cubic yards of solid gold (booty^2). You must cross Roads, Fields, and Mountains to get to your destination.
It takes you 1 hour to move across a square of Road, 2 hours to move across a square of Field, 3 hours to move across a square of Mountain. A castle or tower takes up the entire square, so once you reach the square, you are considered to be in the tower or castle. You obviously want to travel from your Castle to the Princess' Tower as quickly as possible, so which route should you take? Valid movements on this terrain are Up, Left, Right, and Down.

Input
Each test case will be an N x N map represented as follows: The first line of each test case will contain an integer representing N. The next N lines will contain a string of N characters, R for a Road, F for a Field, M for a Mountain. Somewhere on the map will be the starting square, C for your Castle, and the ending square, T for the Princess' Tower.

Each test case is separated with an asterisk (*).

Output
An integer representing how many hours it took you to make your way from the Castle to the Tower along the fastest route.

Input File: C:\DKC3\BootyIn.txt
Output File: C:\DKC3\BootyOut.txt

Examples:

Input:

```
7
RTFMMMF
FMFMMMM
RRFFMRR
FRFFFMR
FRMMFFR
FFMMMMR
RRRCRRR
*
5
MMMMC
FFMMM
RTFMM
FFMFR
RRRRR
*
```

Output:

```
10
11
```

3. Tunnels (75 pts)

Curses! A debonair spy has somehow escaped from your elaborate deathtrap, overpowered your guards, and stolen your secret world domination plans. Now he is running loose in your volcano base, risking your entire evil operation. He must be stopped before he escapes!

Fortunately, you are watching the spy's progress from your secret control room, and you have planned for just such an eventuality. Your base consists of a complicated network of rooms connected by non-intersecting tunnels. Every room has a closed-circuit camera in it (allowing you to track the spy wherever he goes), and every tunnel has a small explosive charge in it, powerful enough to permanently collapse it. The spy is too quick to be caught in a collapse, so you'll have to strategically collapse tunnels to prevent him from traveling from his initial room to the outside of your base.

Damage to your base will be expensive to repair, so you'd like to ruin as few tunnels as possible. Find a strategy that minimizes the number of tunnels you'll need to collapse, no matter how clever the spy is. To be safe, you'll have to assume that the spy knows all about your tunnel system. Your main advantage is the fact that you can collapse tunnels whenever you like, based on your observations as the spy moves through the tunnels.

Input

The input consists of several test cases. Each test case begins with a line containing integers R ($1 \le R \le 50$) and T ($1 \le T \le 1000$), which are the number of rooms and tunnels in your base respectively. Rooms are numbered from 1 to R. T lines follow, each with two integers x, y ($0 \le x,y \le R$), which are the room numbers on either end of a tunnel; a 0 indicates that the tunnel connects to the outside. More than one tunnel may connect a pair of rooms.
The spy always starts out in room 1. Input is terminated by a line containing two zeros.

Output

For each test case, print a line containing the test case number (beginning with 1) followed by the minimum number of tunnels that must be collapsed, in the worst case.

Input File: C:\DKC3\TunnelsIn.txt
Output File: C:\DKC3\TunnelsOut.txt

Examples:

Input:	Output:
4 6 1 2 1 3 2 4 3 4 4 0 4 0 4 6 1 2 1 3 1 4 2 0 3 0 4 0 0 0	Case 1: 2 Case 2: 2

4. Kissin' Cousins (75 pts)

The Oxford English Dictionary defines cousin as follows:
cous'in (k\O(u,ù)zn), n. (Also first cousin) child of one's uncle or aunt; my second (third...) cousin, my parent's first (second...) cousin's child; my first cousin once (twice...) removed, my first cousin's child (grandchild...), also my parent's (grandparent's...) first cousin.

Put more precisely, any two persons whose closest common ancestor is (m+1) generations away from one person and (m+1)+n generations away from the other are mth cousins nce removed. Normally, m >= 1 and n >= 0, but being used to computers counting from 0, in this problem we require m >= 0 and n >= 0. This extends the normal definition so that siblings are zeroth cousins. We write such a relationship as cousin-m-n.

If one of the persons is an ancestor of the other, p generations away where p >= 1, they have a relationship descendant-p.

A relationship cousin-m1-n1 is closer than a relationship cousin-m2-n2 if m1 < m2 or (m1 = m2 and n1 < n2). A relationship descendant-p1 is closer than a relationship descendant-p2 if p1 < p2. A descendant-prelationship is always closer than a cousin-m-n relationship.

Write a program that accepts definitions of simple relationships between individuals and displays the closest cousin or descendant relationship, if any, which exists between arbitrary pairs of individuals.

Input

Each line in the input begins with one of the characters '#', 'R', 'F' or '*'.
'#' lines are comments. Ignore them.

'R' lines direct your program to record a relationship between two different individuals. The first 5 characters following the 'R' constitute the name of the first person; the next 5 characters constitute the name of the second. Case is significant. Following the names, possibly separated from them by blanks, is a non-negative integer, k, defining the relationship. If k is 0, then the named individuals are siblings. If k is 1, then the first named person is a child of the second. If k is 2, then the first named person is a grandchild of the second, and so forth. Ignore anything on the line following the integer.

'F' lines are queries; your program is to find the closest relationship, if any, which exists between the two different persons whose 5 character names follow the 'F'. Ignore anything on the line following the second name. A query should be answered only with regard to 'R' lines which precede the query in the input.

There will be one '*' line to mark the end of a test case.

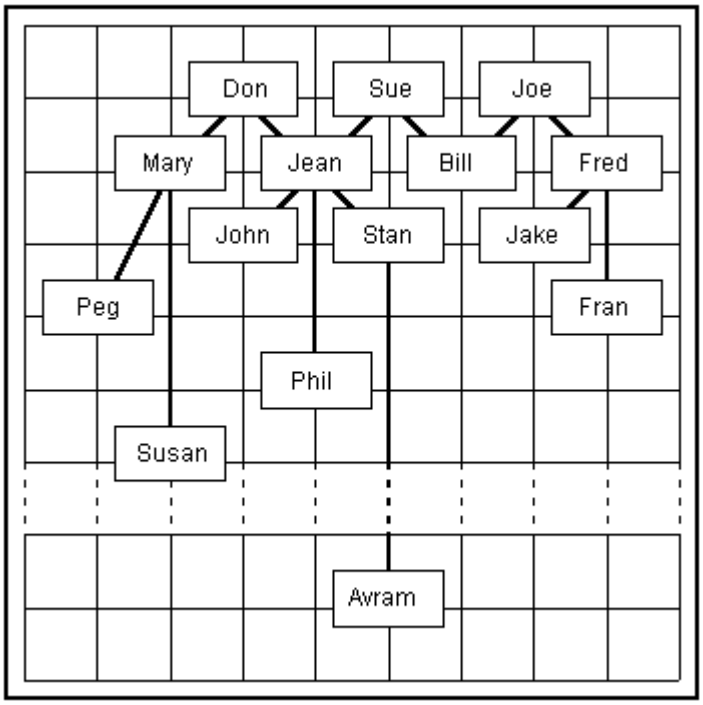
Output

For each 'F' line, your program is to report the closest relationship that exists between the two persons named *aaaaa* and *bbbbb* in one of the following formats:
aaaaa and *bbbbb* are descendant-*p*.
aaaaa and *bbbbb* are cousin-*m*-*n*.
with *m*, *n* and *p* replaced by integers calculated as defined above. If no relationship exists between the pair, your program is to output the following:
aaaaa and *bbbbb* are not related.

Assumption:
A person is not an ancestor of himself/herself.

Input File: C:\DKC3\KissinIn.txt
Output File: C:\DKC3\KissinOut.txt

Diagram of the relationships from example 1 below.



Input:

```
# A Comment!
RFred Joe  1
RFran Fred 2
RJake Fred 1
RBill Joe  1
RBill Sue  1
RJean Sue  1
RJean Don  1
RPhil Jean 3
RStan Jean 1
RJohn Jean 1
RMary Don  1
RSusanMary 4
RPeg  Mary 2
FFred Joe
FJean Jake
FPhil Bill
FPhil Susan
FJake Bill
FDon  Sue
FStan John
FPeg  John
FJean Susan
```

Output:

```
Jean  and Jake  are not related.
Phil  and Bill  are cousin-0-3.
Phil  and Susan are cousin-3-1.
Jake  and Bill  are cousin-0-1.
Don   and Sue   are not related.
Stan  and John  are cousin-0-0.
Peg   and John  are cousin-1-1.
Jean  and Susan are cousin-0-4.
Fran  and Peg   are not related.
John  and Avram are not related.
John  and Avram are cousin-0-99.
Avram and Phil  are cousin-2-97.
```

FFran Peg
FJohn Avram
RAvramStan 99
FJohn Avram
FAvramPhil
*