20th Annual Denison Spring Programming Contest
28 February 2009

Rules:

1. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
2. All programs will be re-compiled prior to testing with the judges' data.
3. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed.
4. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
5. All communication with the judges will be handled by the PC^2 environment.
6. The allowed programming languages are C, C++ and Java.
7. Judges' decisions are to be considered final.
8. There are **six** questions to be completed in **four hours**.
9. **Important:** No extraneous whitespace should appear in your output. Specifically, there should be no blank lines, unless specifically called for. A line should *never* end with whitespace. Do not use tabs in your output, only spaces. Data fields in an output line should be separated by a single whitespace, unless specified otherwise. Any deviations to these guidelines will result in a "Format Error" from the judges.

Problem A: Fibonacci-like

In mathematics, there are several "necklace" problems, one of which is the following: assume you start with two digits a and b, add them, then take the last digit to get a new number c. Then repeat the process. This is like a Fibonacci sequence, only we're just dealing with the last digits. Amazingly, perhaps, this sequence always returns to the original two numbers. The problem is, how many steps does it take to get back to the original a and b (thus completing the necklace)? For example, if we start with 1 and 1, the sequence looks like the following:

1 1 2 3 5 8 3 1 4 5 9 4 3 7 0 7 7 4 1 5 6 1 7 8 5 3 8 1 9 0 9 9 8 7 5 2 7 9 6 5 1 6 7 3 0 3 3 6 9 5 4 9 3 2 5 7 2 9 1 0 1 1

This takes a total of 60 steps altogether. (We count the initial pair, but not the pair when they repeat.) Obviously, this can be done in any base, not just base 10. That is, given a base r, you start with a pair of integers a and b, both less than r, start building the sequence base r, and find how many steps it takes to return to the original a and b. (Again: this will always happen.)

Input

Input for each test case will be three positive integers, a b r, on a line. Both a and b will be less than r, which will be no larger than 100. You will determine the number of steps to complete the necklace when starting with a and b and building the necklace using base r, as described above. The last line, which should not be processed, will have three zeros.

Output

Each test case should produce one line containing the number of steps required to return to the original two numbers.

Sample Input

```
1 1 10
2 3 12
4 5 100
4 8 9
0 0 0
```

Sample Output

```
60
24
300
24
```

Problem B: High Altitude Runner

Emma is a long distance runner and is in Colorado for a bit of altitude training. She has a digitized map of the region, showing the average altitude for each sector on the map. She wants to plan a north-south path so that she maximizes her altitude training — that is, a path where the sum of the altitudes is maximal. The sum of the altitudes of the sectors visited on the path will be the value of the path. Now since she wants to be generally traveling north-to-south, whenever she is in a sector, the next sector visited will be either to the southwest, south, or southeast. She also needs to stay on the map, not wanting to get lost. Emma can start at any point on the north border and end at any point on the south border. Your job will be to find the path (or rather just the value of the path) Emma desires.

Input

The first line of input will be n, indicating the number of test cases. Each test case will have a line containing m ($m \leq 100$) followed by m lines, each containing m positive integers giving the altitudes of grid sectors, west-to-east. The first line gives the northernmost west-to-east data, followed by the next row of data to the south, etc. Altitudes will never exceed 10,000.

Output

Each test case should produce one line of output of the form

`Emma's highest valued path for case` c `is` v`.`

where c is the case number and p is the sum of the maximal north-south path.

Sample Input

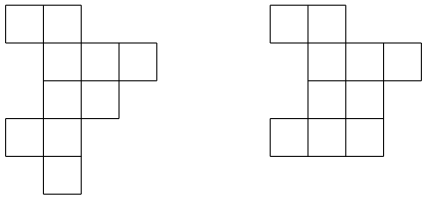
```
1
10
2 3 4 5 6 7 8 7 6 5
2 2 2 3 3 4 4 5 5 6
3 3 4 4 5 6 7 6 5 4
2 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3
3 3 4 4 5 4 3 2 1 1
1 2 3 4 3 2 2 3 9 1
1 2 3 2 3 4 3 8 4 1
2 2 2 2 2 2 6 4 3 3
2 2 2 2 2 2 2 2 2 4
```

Sample Output

```
Emma's highest valued path for case 1 is 53.
```

Problem C: Dominoes

The problem here is a simple one: given a pattern of squares, can you cover the pattern with non-overlapping dominoes? Two patterns are shown below. The left pattern cannot be covered with dominoes while the right pattern can.



Here, the patterns won't be too large: they will all be contained in a 5-by-5 grid.

Input

The first line will be the integer n indicating n test cases follow. Input for each test case will be 5 lines containing five `0`'s or `1`'s (with no intervening spaces), indicating the pattern: a `0` is an"open" square (one not to be covered) and a `1` is a square to be covered by a domino.

Output

Each test case should produce one line of output of the form

`Case` k`: YES.`

or

`Case` k`: NO.`

where k is the test case number.

Sample Input

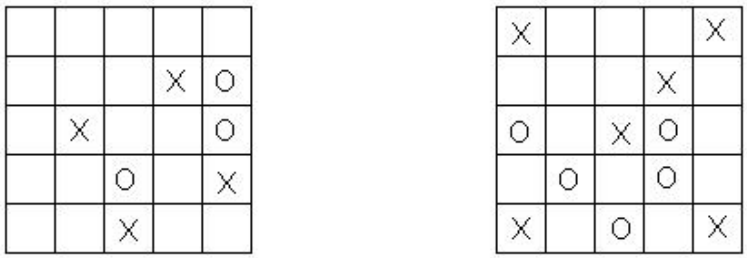
```
2
11000
01110
01100
11000
01000
11000
01110
01100
11100
00000
```

Sample Output

```
Case 1: NO.
Case 2: YES.
```

Problem D: Be a Square

Here's an interesting variation on tic-tac-toe: You have an n-by-n grid of squares where players alternate placing `X`'s and `O`'s on squares that have not yet been marked. A player wins when they've marked the four corners of a square. The interesting part is the sides of the square need not be horizontal and vertical. The two boards below show a winning configuration for `X`, the first after 7 moves, and the second after 11 moves (assuming `X` moves first).



You'll be given n^2 moves and should determine who wins the game on what move if played on an n-by-n board. (Simply ignore the moves listed after the winning move.) If there is no winning move, then it's a cat's game (that is, a tie).

Input

Input for each test case will start with a line containing n, indicating the size of the board. ($n = 0$ indicates end of input and should not generate any output.) n will be greater than 3, but no larger than 10. There then follows (on the next line) n^2 pairs of values, giving the alternating `X` and `O` moves (`X` will always move first). A move is a row/column pair. You may assume that the rows and columns of the board are indexed starting at 1 and the moves cover the board.

Output

Each test case should produce one line of output of the form

`Game` g `is won by` p `on move` m`.`

or

`Game` g `ends in a cat's game.`

accordingly, where g is the game number, p is either `X` or `O`, and m is the move that gives p the win.

Sample Input

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

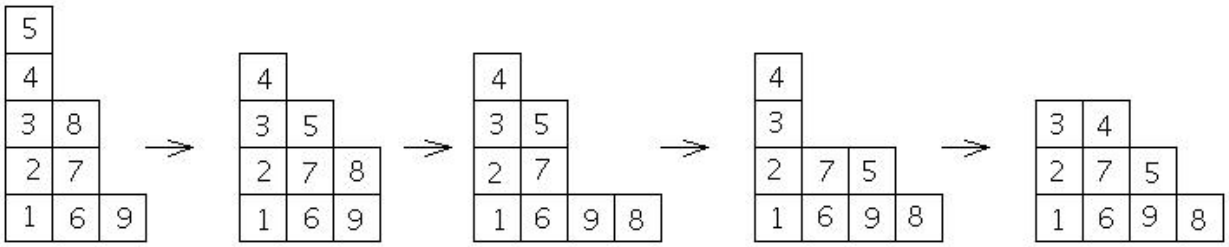
Sample Output

```
Game 1 is won by X on move 7.
Game 2 is won by X on move 11.
```

Problem E: Trucks and Boxes

Emma works in a warehouse where boxes are stacked up in sometimes very uneven piles. One of her jobs is to rearrange the boxes so to make the piles a bit more stable. She uses the best tool she has available for the job — a truck. Her technique is to ram the truck into the left end of the pile of boxes. When she does so, any box that can fall to the right (that is, the box is at the top of its pile and there is no box to its immediate right at the very next level down) does so. All the boxes that can fall, do so simultaneously and depend only on if there is space to drop at the time of ramming. Emma then backs up and rams again. She repeats until there are no more boxes to fall, then breaks for a coffee, hoping that there was nothing fragile in those boxes. (Actually, that doesn't really bother her so much.)

Below is a small example of Emma's handiwork. Notice the original configuration on the left. The first ram causes boxes 5 and 8 to fall. The second ram causes box 8 to fall (but not box 5, since 8 was in the way). The third ram causes box 5 to fall (but not box 4). Finally, box 4 falls to form the stable configuration.



Input

The first line of input will indicate the number of test cases. Input for each test case will be on a single line of the following form:

$n\ c_0\ c_2\ \cdots\ c_{n-1}$

where n is the number of piles of boxes (all pushed together) and c_i is the number of boxes in position i. $c_i > 0$ for all i. The boxes will be numbered starting at 1 starting at the leftmost (first) position and counting up, as shown in the example. The total number of boxes will never exceed 100.

Output

Each test case should produce one line of output of the form

`Case` p`:`$B_1\ B_2\ \cdots\ B_b$`.`

where p is the test case number and $B_1\ B_2\ \cdots\ B_b$ are the block numbers as read from bottom-to-top, left-to-right.

Sample Input

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

Sample Output

```
Case 1: 1 2 3 6 7 4 9 5 8.
Case 2: 1 2 5 3 4.
```

Problem F: Niminum

A number game (called *niminum*) is played between two players as follows: Starting with the positive integers, player A picks a number from the set. That number and all multiples are removed from play. Player B then does the same, picking from the numbers remaining in play, with the number picked and all multiples removed. In addition, enough numbers are removed so that the sum of any two removed numbers is also removed. (That is, the sum of any two removed numbers is removed.) For example, if A picks 5 and B then picks 3, then all multiples of 5 are removed on A's turn. On B's turn, all multiples of 3 plus every number that is the sum of a multiple of 3 and a multiple of 5 is removed. For example, 11(among many other numbers) is removed in the second turn. At this stage the numbers remaining are 1, 2, 4, and 7. The loser is the one who makes the last move — which is obviously picking 1.

You will be given a stage of the game where only a finite number of integers remain and a series of legal moves, and asked what the remaining integers are. You are going to simulate two turns of the game. A will make a pick, then B will make a pick. (They both will be legal.) You will simply print out the integers left in the set.

Input

Input for each test case will consist of one line of the form

$n\ a_1\ a_2\ \cdots\ a_n\ A\ B$

where $a_1 < a_2 < \cdots < a_n$ represent the numbers remaining in play ($a_n \leq 100$ and $n \leq 50$), A is player A's move and B is player B's move. Both A and B are numbers that are in play.

The test cases will be followed by a line containing only 0.

Output

Each test case should produce one line of output of the form

`The numbers remaining in case` c `are` $b_1\ b_2\ \cdots\ b_m$`.`

where c is the test case number, and $b_1\ b_2\ \cdots\ b_m$ are the integers remaining in play, in increasing order, if there are any numbers left, or

`There are no numbers left in case` c`.`

if the game ends after player B's move. (Note: since B's move will always be legal — that is, not removed as a result of A's move — the only way there will be no numbers left is for B to make the last (losing) move of the game.)

Sample Input

```
7 1 2 3 4 6 9 11 6 3
16 1 2 3 4 5 6 8 9 10 11 13 17 18 20 23 25 9 3
0
```

Sample Output

```
The numbers remaining in case 1 are 1 2 4.
The numbers remaining in case 2 are 1 2 4 5 8 11.
```