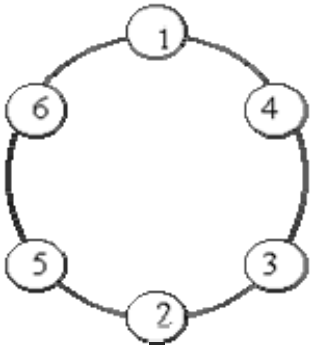
Problem A
Intriguing Player Chain

Input : Standard Input.
Output : Standard Output.

Players in the Hermes sports complex are wearing jerseys imprinted with natural numbers. There are even number of players. During their practice session they were asked to form a circular chain. The players have done so but with a mathematical charm. The instructor Mr. Ramanujan, after taking a round around the player chain, noticed that the sum of any two adjacent jerseys had an interesting property (***adjacency constraint***). That number is divisible by itself and one only. This found to be true both in a clockwise or in an anticlockwise direction. A 6 player ring is illustrated below.



Your job is to automate Mr. Ramanujan's brain to see all possible such patterns, given any n (Maximum value of n is 16). Captain has jersey #1 and hence obviously the point of beginning. Print all possible formations in ascending numeric ordering e.g. 1 2 5 comes before 1 2 6. Every formation must begin with captain's jersey number. Give priority to clockwise formations i.e. print patterns in clockwise direction and then all patterns of anticlockwise direction. There are no missing numbers on the jerseys. E.g. if the number of players is 6, the jerseys are numbered 1, 2…6.

Input (Standard Input)

The first line of the input represents the number of test cases k (at most 5).
The next k lines represent the number of players n ($0 < n <= 16$).

Output (Standard Output)

The output format is shown as sample below. Each row represents a possible formation satisfying the above adjacency constraint. For each test case, the first line should be the test case number followed by the arranged patterns, one pattern per row. There should be an empty line separating each test case.

Sample Input

```
2
6
8
```

Sample Output

```
1
1 4 3 2 5 6
1 6 5 2 3 4

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

Problem B
Longest Run on a Snowboard

Input: standard input
Output: standard output

Mickie likes snowboarding. That's not very surprising, since snowboarding is really great. The bad thing is that in order to gain speed, the area must slope downwards. Another disadvantage is that when you've reached the bottom of the hill you have to walk up again or wait for the ski-lift.

Mickie and his friend Minnie are on a snowboarding adventure. Minnie offers to pay Mickie in thousands of dollars equivalent to the number of points he covers. But she poses a constraint: during his ride he should not choose a point whose height value is co-prime to the current point's height. (Two integers are co-prime if they do not have a common factor e.g. 14 and 15.)

Mickie has urgently called you over phone and pleads you to help him in his challenge. He would like to know how long the longest run in an area is. That area is given by a grid of numbers, defining the heights at those points. Look at this example:

```
 7  2  3  4 5
36 37 38 34 6
33 44 46 40 7
24 43 42 41 8
35 32 31 30 9
```

One can slide down from one point to a connected other one if and only if the height decreases and the heights are not co-prime. One point is connected to another if it's at left, at right, above or below it. In the sample map, a possible slide would be **36-33-24** (start at **36**, end at **24**). Of course if you would go **46-44-33-24 or 46-38-34-4 or 46-40-34-4**, it would be a much longer run. If there are more than one run of equal length, pick the one where the first different point comes earlier in the grid (minimum in x-co-ordinate first, and then in y, assuming that the top left corner is (0,0)). In the case above, we will pick **46-38-34-4**.

Input (Standard Input)

All input comes from standard input. The first line contains the number of test cases **N**. Each test case starts with a line containing the name (it's a single string), the number of rows **R** and the number of columns **C**. After that follow **R** lines with **C** numbers each, defining the heights. **R** and **C** won't be bigger than **100**, **N** not bigger than **15** and the heights are always in the range from **2** to **100**.

Output (Standard Output)

For each test case, print a line to standard output containing the name of the area, a colon, a space and the length of the longest run (maximum points covered) one can slide down in that area. The path taken by Mickie is printed in the next line, each point separated by a blank space from the next.

Sample Input

```
2
disney 5 5
96 57 56 41 93
87 58 33 83 77
49 52 39 76 66
47 79 36 31 43
3 12 27 29 2
midas 4 3
87 86 5
71 72 57
64 51 3
45 31 22
```

Sample Output

```
disney: 9
96 87 58 52 39 36 27 12 3
midas: 4
86 72 57 3
```

Problem C
Operator Jumble

Input: standard input
Output: standard output

A popular word puzzle asks you to punctuate the following word sequence so that it makes sense.

"James while John had had had had had had had had had had had a better effect on the teacher."

The answer is:James**,** while John had had "had"**,** had had "had had"**;** "had had" had had a better effect on the teacher**.**

Similarly, number sequences that don't make sense by themselves can be made into correct equations by the addition of appropriate arithmetic operators and the = sign.

For example,

5 7 4 = 3 is more meaningful when written as

5 + 7 / 4 = 3.

We would like you to write a program to insert appropriate operators (+, -, * and /) into the integer sequences supplied by us so that each sequence is transformed into a mathematically correct equation.

Each number in the input sequence must be used exactly once, but each operator may be used zero to many times. The expression should be read from left to right, with the output of the first operation being input of the second and so on, to calculate the target number. It is possible that no expression can generate the target number. It is also possible that many expressions can generate the target number. In this case, the 'correct' solution will be the one where the operator sequence is in 'alphabetic' order, where + comes first, then -, * and / in that order. E.g. 1 + 1 * 1 comes before 1 + 1 / 1.

There are three restrictions on the composition of the mathematical expression:

- the numbers in the expression must appear in the same order as they appear in the input.
- you are only allowed to use / in the expression when the result of the / operator will give a remainder of zero. Division by 0 is not allowed.
- you are only allowed to use an operator in the expression if its result after applying that operator is an integer from (**-32000** .. **+32000**). The outer limits of this interval are not allowed.

Input (Standard Input)

The input has multiple test cases. The first line contains the number of test cases **n**.

Each subsequent line contains the number of integers in the sequence **p**, followed by **p** non-negative integers, followed by the target integer. Each integer is separated by a space from the next. Note that **0 < p <= 100**. There may be duplicate numbers in the sequence. But all the numbers are less than **32000**.

Output (Standard Output)

The output should contain an expression, including all **p** numbers and **(p-1)** operators plus the equals sign and the target. Do not include spaces in your expression. If there is no expression possible output **"NO EXPRESSION"** (without the quotes).

Sample Input

```
3
3 5 7 4 3
2 1 1 2000
5 12 2 5 1 2 4
```

Sample Output

```
5+7/4=3
NO EXPRESSION
12+2-5-1/2=4
```

Problem D

Mahabharat – A Story in disguise

Input: standard input
Output: standard output

One of the greatest epics in Indian history is Mahabharat. In this epic, Shakuni plays a key role by intimidating Dhuryodhana to go against the Pandavas resulting in the Great War. The story that still remains hidden is what causes Shakuni to intimidate Dhuryodhana resulting the down fall of Dhuryodhana. Shakuni himself unfolds the story for us:

I was born along with one hundred brothers and enjoyed a huge amount of wealth. Dhuryodhana had an eye over my wealth and wanted to capture it, so he imprisoned me and my brothers. He provided food and water sufficient for only one person. Since, I was the most intelligent of the lot and was keen to take revenge on him, I decided to make sure that I got the food needed to survive. So I framed an idea to get the food.

The idea was to form a circle consisting of me and my brothers and then eliminating the **m**th person from the circle stating he will not be eligible for the food. The last person still left in the circle would get the food and water. I always made sure that I got the food and water. Also, I wanted to make sure that none of my brothers recognize the pattern of elimination, so I kept changing the value of **m** every day.

Assume yourself to be the modern-world Shakuni equipped with computers, write a program to obtain the food for yourself given the number of brothers (n) and the elimination number (m).

Input(Standard Input)

The standard input consists of k(at most 5) test cases. First line of the input is the number of test cases. The next k lines each has two numbers n (0<=32000) and m (0<=32000) separated by a space, each corresponding to a single test case. n is the number of brothers and m is the elimination value. The first position is numbered "1".

Output(Standard Output)

For each test case, print the position you should take to obtain the food. E.g. for the 23rd position, output the number 23. You should have k lines of output, and no blank lines.

Sample Input

```
2
41 3
1000 55
```

Sample Output

```
31
63
```

Problem E
Jungle Safari

Input: Standard Input.
Output: Standard Output.

In this problem, you will compute how many water packets you need to purchase for a trip across the jungle on foot.

At your starting location, you can purchase water packets at the general store and you can buy an unlimited amount of fruits available at the general store. The jungle contains trees with edible fruits only in specific locations. At each such location, you can collect as many fruits as you like and you can store water packets for later use. You can also purchase water packets for later use from the store. You will be given the coordinates of the store (starting location), location of trees that have edible fruits, and your destination in a two-dimensional coordinate system where the unit distance is one mile.

For each mile that you walk, you must consume one packet of water and one fruit.
Assume that these supplies are consumed continuously, so if you walk for a partial mile you will consume partial units of water and fruit. You are unable to walk unless you have supplies of both water and fruit. You must consume the supplies while you are walking, not while you are resting under the tree. Of course, there is a limit to the total amount of water packets and fruits that you can carry. This limit is expressed as your carrying capacity in total units (1 water packet or 1 fruit is 1 unit). At no time can the sum of the water packets and the fruits that you are carrying exceed this capacity.

You must compute minimum number of water packets you need to purchase at the store in order to reach the destination.
You need not have any water packet or fruit left when you arrive at the destination. Since the general store has only one million water packets available, the amount of water you should buy will be an integer greater than zero and less than or equal to one million.

Input (Standard Input)

The first line of input in each trial data set contains n (2 <n< 20), which is the total number of significant locations in the jungle, followed by an integer that is your total carrying capacity in units of fruits and water packets. The next n lines contain pairs of integers separated by a space that represent the coordinates of the n significant locations. The first significant location is the store (starting point), where water packets must be purchased; the last significant location is the destination; and the intervening significant locations (if any) are trees with edible fruits. You need not visit any tree with edible fruits unless you find it helpful in reaching your destination, and you need not visit these trees in any particular order.
Each test case / trial is terminated by a pair of zeroes separated by a space, on a line by itself

Output (Standard Output)

For each trial, print "Trial #trial number:" followed by an integer that represents the minimum number of water packets needed for your journey. Use the format shown in the example. If you cannot make it to the destination under the given conditions, print the trial number followed by the word "Impossible" .Trailing newline characters at the very end of the output will be ignored.

Place a blank line after the output of each test case.

Sample Input

```
4 100
10 -20
-10 5
30 15
15 35
2 100
0 0
100 100
0 0
```

Sample Output

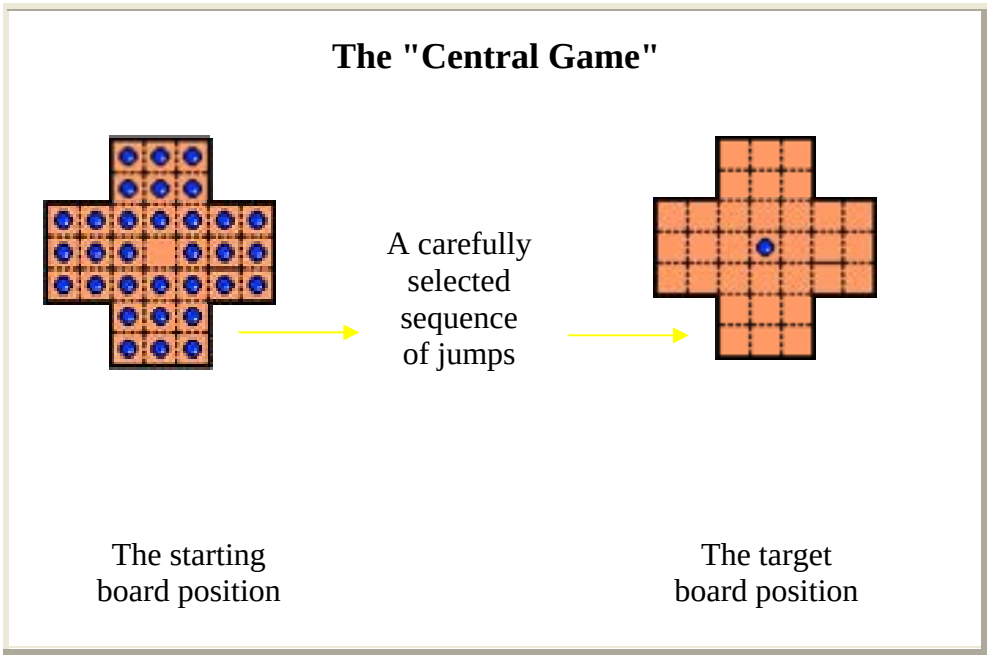
```
Trial 1: 136

Trial 2: Impossible
```

Problem F
Peg-Solitaire

Input : Standard Input.
Output : Standard Output.

Peg solitaire is a classical puzzle (originating in France in the late 17th century) commonly played on a 33-hole cross-shaped board (also called "the English Board") .In England it is known as "Solitaire", but in the U.S. most people think of this as a card game so it is usually called "Peg Solitaire". Other people know it as "Hi-Q" because a popular version of the game sells under that trade name.



The basic game consists of a cross-shaped board, usually made of wood, together with a set of 32 pegs. To start the game, one fills the board with pegs except for the central hole. A jump consists of jumping one peg over another into an empty spot in the board, removing the peg jumped over from the board. Diagonal jumps are not allowed (in the standard version of the game). The goal is to choose a sequence of jumps and finish with as few pegs as possible, ideally a single peg in the center

Starting with the centermost hole open, players move the pegs by jumping one peg over another, either in a horizontal or vertical direction and removing each peg that is jumped over. The objective for players is to remove as many pegs from the board as possible. This problem involves writing a program that will automatically play Peg-Solitaire so that we can investigate how the game might unfold based on various opening arrangements of pegs.

Problem Statement: There is a peg board with the following shape and with holes numbered from 1 to 33 as follows:

```
                    1     2     3
                    4     5     6
    7     8     9    10    11    12    13
   14    15    16    17    18    19    20
   21    22    23    24    25    26    27
                   28    29    30
                   31    32    33
```

An instance of the game begins with some holes having pegs in them and the rest of the holes being empty. The game proceeds by jumping one peg over another, either horizontally or vertically, with the peg that is jumping landing in an empty hole, and the peg being jumped over being removed from the board. For example, if 9 is empty and 10 and 11 are not, then the peg in 11 can be "moved" to 9 with the peg in 10 being removed. After this move, 10 and 11 would both be empty but 9 would have a peg in it.

Given a specific board configuration your program will pick and model a specific move, over and over, until no more moves are available. Your program will then report the number of holes that still have pegs in them and their positions on the board. At any point during the game there may be more than one possible move available. In such a case always model the move with the target hole of the moving peg as large as possible. If there is more than one move available to the largest possible target hole, then choose from those moves the one with the larger source hole.

For example, if the board looks like this, with X representing a peg and O representing a hole:

```
                    O     O     O
                    O     O     O
    O     O     O     X     O     X     O
    O     O     O     X     O     X     O
    O     O     O     O     X     O     O
                    O     O     O
                    O     O     O
```

then the following jumps would be made:
1: from 12 over 19 to 26 (26, 24, and 5 are the only possible targets and 26 is the largest),
2:from 25 to 27 over 26 (5 ,24 and 27 are the only possible targets and 27 is the largest)
3: from 10 to 24 over 17 (24 > 5), and two pegs would be left, one in hole 24 and one in hole 27.
Thus 2 pegs at holes 24 and 27 would be reported as the result for this instance.

Input (Standard Input)

The first line contains an integer N between 1 and 10 describing how many instances of the game are represented. N lines follow describing N instances of the game by listing the hole that have pegs in the beginning. Holes are listed in ascending order separated by a single space. A 0 will indicate the end of each sequence of unique number between 1 and 33 that represents an instance of the game.

Output (Standard Output)

There should be N+2 lines of output. The first line of output will read Peg-Solitaire OUTPUT. There will then be one line of output for each instance of the game, reporting the number of the holes that still have pegs in them, then a single space, and then their positions for the final configuration of that instance in ascending order separated by single spaces. The final line of output should read END OF OUTPUT.

Sample Input

2
1 4 7 8 10 18 0
1 4 7 8 10 0

Sample Output

Peg-Solitaire OUTPUT
4 1 10 16 18
3 1 10 16
END OF OUTPUT

Problem G
Rat Attack

Input: standard input
Output: standard output

Baaaam! Another deadly gas bomb explodes in Manhattan's underworld. Rats have taken over the sewage and the city council is doing everything to get the rat population under control.

As you know, Manhattan is organized in a regular fashion with streets and avenues arranged like a rectangular grid. Waste water drains run beneath the streets in the same arrangement and the rats have always set up their nests below street intersections. The only viable method to extinguish them is to use gas bombs like the one which has just exploded. However, gas bombs are not only dangerous for rats. The skyscrapers above the explosion point have to be evacuated in advance and so the point of rat attack must be chosen very carefully. The gas bombs used are built by a company called American Catastrophe Management (ACM) and they are sold under the heading of "smart rat gas". They are smart because — when fired — the gas spreads in a rectangular fashion through the under street canals. The strength of a gas bomb is given by a number d which specifies the rectangular "radius" of the gas diffusion area. For example, Figure shows what happens when a bomb with d = 1 explodes.

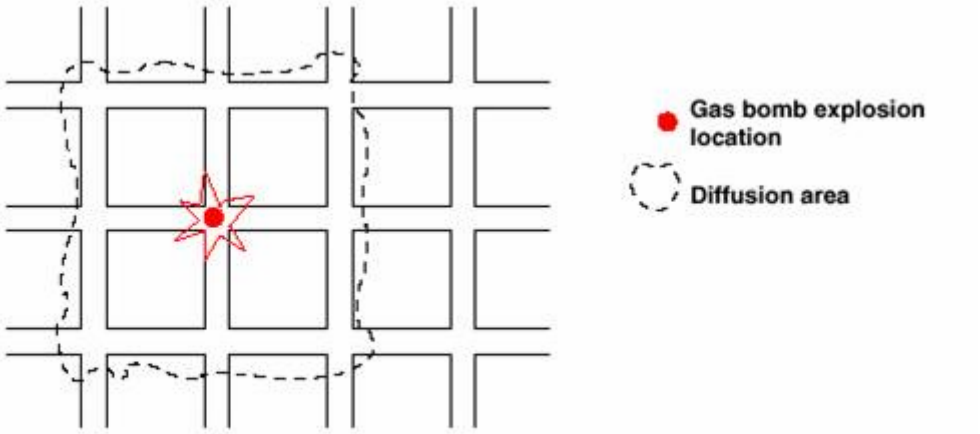


Fig: Rectangular diffusion area of gas bomb

The Problem

The area of interest consists of a discrete grid of 1025 × 1025 intersection points. Rat exterminator scouts have given a detailed report on where rat populations of different sizes have built their nests. You are given **m** gas bombs with strengths d1,d2,d3.....dm respectively and your task is to find explosion locations for these gas bombs which extinguish the largest number of rats.

The best position is determined by the following criteria:

• The sum of all rat population sizes within the diffusion area of the gas bomb (given by d) is maximal.

• If there are more than one of these best positions then the location with the “minimal” position will be chosen. Positions are ordered first by their x coordinate and second by their y coordinate.

Formally, given a location (x1, y1) on the grid, a point (x2, y2) is within the diffusion area of a gas bomb with strength d if the following equation holds:

max (abs(x2 - x1), abs (y2 - y1)) <= d

The positions of the bombs must be disjoint . The optimal number (for the same number of rat deaths, the fewer the bombs used the better) of bombs must be used so that the maximum number of rats are killed.

Input (Standard Input)

The first line contains the number of scenarios in the input (from standard input). For each scenario the first line contains the number of bombs m(1<=m<=50) and the number n (1 <= n <= 20000) of rat populations separated by a blank space. The second line consists of strengths of each bomb separated by a blank space. Then for every rat population there follows a line containing three integers separated by spaces, for the position (x, y) and "size" i of the population (1 <= i <= 255). It is guaranteed that position coordinates are valid (i.e., in the range between 0 and 1024) and no position is given more than once.

Output (Standard Output)

For every scenario print a line to standard output containing the x and y coordinate of the chosen location for the gas bombs, followed by the sum of the rat population sizes which will be extinguished by each bomb used. The three numbers must be separated by single spaces. Each scenario must be separated by a blank line.

Sample Input

```
2
1 2
1
4 4 10
6 6 20
2 2
20 10
100 1010 20
20 10 200
```

Sample Output

```
5 5 30

0 0 200
90 1000 20
```