A: We Only Stop For The Best

You are travelling down an ocean coast from north to south. Each place that you pass along the way has many things to recommend it to you, but you have a limited amount of time to spend, and each stop costs you a day of your travel time. It also takes a day to get between each place whether you stop or not. Furthermore, each place changes day to day, with different events and weather patterns that affect how much enjoyment you will get from the place. If you stay for more than one day in the same place, your enjoyment is the maximum of that place's enjoyment over each of the different days. Travelling can be fun, but too much of it is not, and each day without a stop (after the initial day in a sequence of days without stops) provides a penalty to your total enjoyment. You must reach the end of the trip (last place) by the end of your last day, and you can only travel forward (north to south). Completing early allows you extra days at the last place, but only the most enjoyable of these days should be counted, as with other stops.

Input

The first line of the input consists of the number of cases k. For each case, it's first line consists of n, the number of days that you have in total, m, the number of places, and p, the travel penalty (all expressed as positive integers). This is followed by m lines, one for each place in sequence, and each line consists of n integers, indicating the enjoyment that you could get from stopping in that place for each of the n days, in order. n and m are each at most 400, and n is at least m.

Output

For each case, print a line giving the highest total enjoyment that can be had from the trip, followed by a colon, then followed by the sequence of stops that should be made, with each stop preceded by a space. The stops are numbered from 1. If more than one sequence of stops produces the same highest total enjoyment, the sequence that has the earliest enjoyment stops, in lexicographic order, should be chosen. For example, (1, 2, 5) should be chosen over (1, 3, 4).

Sample Input	*Sample Output*
2 5 3 1 2 1 5 10 1 1 0 10 6 3 9 8 7 6 10 4 3 5 30 12 100 70 40 10 19 10 10 10 40 100	22: 1 2 3 125: 1 3

B: Waiting at Godot's

In order to make ends meet under the current economic regimen, you find yourself working as a waiter at Godot's, a Manhattan-style Greco-Romanian eatery laid out on an $n \times m$ grid such that each table has co-ordinates (i, j), $0 \leq i \leq n$ and $0 \leq j \leq m$, and the door to the kitchen is at grid-point $(0, 0)$. A set of orders are ready for delivery from the kitchen to their tables. Each order has an associated (x, y) table-position, the weight of the order in Manuels (the standard weight-unit in the fine dining industry), and a tip expected on delivery of the order. The management at Godot's has decreed that in the interest of personal service, at least one and at most three tables can be serviced on any one trip from and back to the kitchen. On any trip you make, you cannot carry orders whose summed Manuel-weight exceeds your Manuel-capacity c; moreover, as you would like to maximize the amount of money you make over your trips (as other waiters are also clamoring to deliver the orders), you must on every trip maximize the Benson Ratio $B = T/D$, where T is the summed tips accumulated on the trip and D is the total distance travelled to deliver the orders on that trip, *e.g.*, from the kitchen to the first table, first table to second table, second table to third table, and back to the kitchen. Note that as tables are laid out on a grid, distances are computed using the city-block metric, *i.e.*, $d((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$.

Write a program which, given your Manuel-capacity c and a set of orders, computes and outputs the delivery-sequence of up to three tables that maximizes the Benson Ratio. As the Benson Ratio of a delivery-sequence is the same for the reversed version of that sequence, *e.g.*, $B([1, 4, 2]) = B([2, 4, 1])$, print the lexicographically smaller sequence as output for convenience, *e.g.*, $[1, 4, 2]$. If there is no valid delivery-sequence, *i.e.*, every possible delivery-sequence has a summed weight that exceeds c, print the message "No valid delivery-sequence".

Input

Your input file will consist of a number of delivery scenarios. The first line in the file gives the number of scenarios in the file. Each scenario with l orders is described by an $(l + 1)$-line block, where the first line in the block gives c and l and each remaining line describes an order with four integers giving the x- and y-coordinates of the table, the weight of the order in Manuels, and the expected tip. You may assume that all input files are formatted correctly, and that each scenario has a unique optimal solution (modulo reversal).

Output

Follow the output format as given in the example. Note that tables are named starting with 1. If there is no valid sequence, print the line:
`Case # n : No valid delivery-sequence`

(Continued on the next page.)

Sample Input

```
2
3 5
1 1 5 30
4 4 3 50
0 2 5 25
0 4 5 30
1 0 3 10
2 2
3 2 5 12
0 3 4 4
```

Sample Output

```
Case # 1 : Maximum Benson Ratio =  5.0 (K -> 5 -> K)
Case # 2 : No valid delivery-sequence
```

C: n-Squared Matrix

Let's, for the sake of argument, use the term "n-squared matrix" for the n-by-n matrix that contains all values in the range $1 \ldots n * n$, entered by row. For example, here is the 3-squared matrix:

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}$$

Note that the values on each diagonal sum to the same value in such a matrix.

Your task is to write a program that does the following:

1. Reads in a single positive integer, and writes out a single positive integer.
2. If the positive integer input, say n, is a perfect square, the positive integer output must be the value which is the sum of the values on either diagonal of the corresponding n-squared matrix.
3. If the positive integer input, say n, is not a perfect square, the positive integer output must be the sum of all the values in the corresponding n-squared matrix.
4. If the input integer is negative, an assumption should be made that its absolute value is the common value of the sum of the two diagonals of an n-squared matrix for some value of n, and it is this value of n that must be output.
5. If the input is 0, then simply stop the execution of the program.

For example, if 4 is input, the output should be 34; if 3 is input the output should be 45, while if -15 is input the output should be 3, which would be the two possible input/output combinations for the above matrix example.

Assume that the single input integer will be left-justified on the single line of input, and the single output value must appear left-justified on its output line.

Input

Input is a sequence of integers, each integer on a line by itself, ending with 0.

Output

Output is a sequence of results, each result on a line by itself. There is no line corresponding to the final 0 in input.

(Continued on the next page.)

Sample Input

```
4
3
-15
0
```

Sample Output

```
34
45
3
```

D: Software Development

We would like to analyze whether some characteristics of the software development process have impact or not on its outcome. We will restrict ourselves to measures that are "categorical" (i.e., the possible values for one measure—or variable—are from a given enumeration of strings). There is no assumption of ordering across the values defined.

Here are examples of some characteristics:

- ProjectSize: small, medium, or large
- DevelopmentApproach: iterative, spiral, waterfall, or RUP

And here are examples of outcome indicators:

- ProductSatisfaction: VerySatisfied, Satisfied, Neutral, NotSatisfied, or CompletelyNotSatisfied
- ScheduleChange: none, few, or often

The data to be analyzed will be one "datapoint" per project, each datapoint containing one value for each of the characteristics and each of the outcome indicators. The analysis itself should detect if some single characteristic or a combination of characteristics leads to "stability" in one outcome indicator (i.e., when taking only the projects with a given characteristic or a given combination of characteristics, for a given outcome indicator, more than 50% of the values reported are one of them in particular). One example could be that having a small project size leads to clients being "Satisfied" with the product most of the time. Another example could be that large waterfall projects typically lead to schedule changes happening often. It should be noted that if for a particular characteristic (or combination of characteristics), only one datapoint could be found, then no analysis should be performed on that case.

Input

The input consists of multiple test cases. For each test case, the input contains two parts. The first one provides the description of the characteristics and the outcome indicators used (see the sample input: the numbers beside "Characteristics" and "OutcomeIndicators" indicate the number of those defined here in the input). The lines for each characteristic and outcome indicator first show its name, followed by the number of possible values, and then the list of all such values. It is assumed that names and values will contain only letters (case sensitive though). The second part of the input starts with the number of datapoints, followed by one line per datapoint (or project). A datapoint line shows the values assigned to each of the characteristics and outcome indicators, in the order that those were presented in the first part of the input. Note that the different pieces of information presented on one line are all separated by a single space.

Output

For simplicity here, the output will contain only statistics on the analysis performed, not the actual result. The output should contain one line per test case. Such line should have two integers

separated by a space: the first one should be the total number of stability relation found, and the second one should be the average (rounded to the nearest integer, with .5 rounding up) of the percentages calculated that were larger than 50%. In the case that no stability relation is found for one test case, the related output should be two zeros.

The output below corresponds to the following analysis results performed internally:

Case 1:

When ProjectSize=small, ProductSatisf = Satisf 100% of the times
When DevelopmentApproach=waterfall, ScheduleChange= often 80% of the times
When ProjectSize=large and DevelopmentApproach=waterfall, ScheduleChange= often 75% of the times

Case 2: no stability relations found

(Continued on the next page.)

Sample Input

```
Characteristics (2):
ProjectSize 3 small medium large
DevelopmentApproach 4 iterative spiral waterfall RUP
OutcomeIndicators (2):
ProductSatisf 5 VerySatisf Satisf Neutral NotSatisf CompletelyNotSatisf
ScheduleChange 3 none few often
10
medium waterfall VerySatisf often
small iterative Satisf none
small spiral Satisf few
small RUP Satisf often
large waterfall Satisf often
large waterfall VerySatisf often
large waterfall NotSatisf often
large waterfall Satisf few
large iterative Neutral few
large spiral VerySatisf none
Characteristics (1):
UIfocused 2 yes no
OutcomeIndicators (1):
OnBudget 2 yes no
3
yes yes
yes no
no yes
```

Sample Output

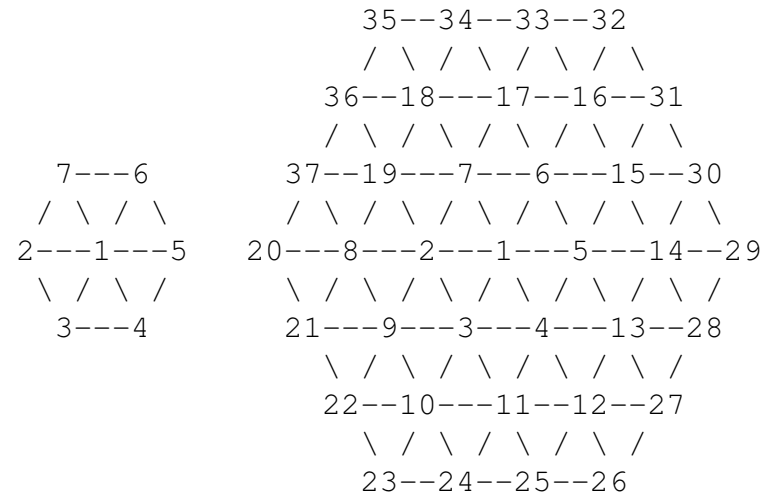
```
3 85
0 0
```

E: Triangle Meshes

We need your help in navigating in hexagonal cut-outs of triangle meshes (see mesh figures below).

In these meshes, each number represents a room and each room has up to 6 neighbouring rooms. Each room in the interior is connected to the 6 rooms around it in the same way that room 1 is connected to the rooms 2–7 in either figure. For example, in the figure on the right, room 2 is connected to rooms 8, 9, 3, 1, 7, and 19. If we denote North, South, East and West by N, S, E, W then each room is connected to those rooms that are W, SW, SE, E, NE, and NW of itself. Note that rooms directly N and S are not connected. E.g. 1 is not connected to 17 or 11. Rooms on the exterior have fewer neighbouring rooms.

All the structures in this problem are hexagonal cut-outs made up of regular concentric hexagonal rings. Two examples are shown below. The figure on the left has a hexagonal ring of numbers 2–7 around the number 1. The figure on the right has three hexagonal rings with the numbers 2–7, 8–19, and 20–37. Larger structures with more than 3 rings may also be used in this problem.



Given a room number, you must find a "fire exit" for the room. A fire exit is the shortest route to get from that room to a room on the exterior of the hexagonal structure. For example, a fire exit for room 1 in the figure on the right is `1*2*8*20`. If there is more than one shortest fire exit, you must report the shortest route that goes to the lower number neighbour first, and if there are two such routes, then the next number should be the lowest, etc; *i.e.,* the smallest sequence in the lexicographic order. For rooms on the outside of a structure the fire exit is just the room number, like for room 32 in the sample input and output.

Input

On any input line, the first value corresponds to the non-negative number of hexagonal rings in the structure, excluding the "trivial" ring with just room 1 in it, and the subsequent (zero or more) numbers on the input line are the rooms for which we want the fire exits in that structure. For

example, in the sample input, the first input line corresponds to a structure with 1 ring (the figure on the left) and asks for the fire exit from room 1. The second line correspones to a structure with 3 rings (the figure on the right) and asks for the fire exits from rooms 4, 1, 32, 17 and 10.

The last line of input will have the number -1.

The input may be free format but the output should be carefully formatted as shown.

You may assume that no structure has more than 10 rings around the 1.

Output

For each line of input, there should be one line of output listing all required fire exits, separated by one space character. No output is produced for the line containing -1.

Sample Input

```
1      1 6
3      4 1 32 17 10
-1
```

Sample Output

```
1*2 6
4*11*24 1*2*8*20 32 17*33 10*22
```

F: Aquarium

The local aquarium wants to reconfigure its displays throughout the aquarium building on a weekly basis. To that end, it is considering buying fish tanks on wheeled platforms that would let the staff move the tanks easily even when filled with water (which will be the normal case). The aquarium is considering tank models with casters as wheels so the staff can turn a tank around on the spot and push the tank in any direction.

The aquarium building has long, straight corridors between rooms that the tanks will need to navigate. Corridors in the building have different widths, so a tank may not fit in every corridor.

The tank manufacturer has several open-top models of tanks, all with a convex footprint (outline when seen from above) and vertical walls.

Your job is to identify, for each tank model, the smallest corridor into which the tank will fit when the tank is filled with water. If a tank model will not fit into any corridor, report "none// as the smallest corridor that it will fit.

Permissible assumptions:

- each tank footprint is described by a convex polygon whose vertices have integer coordinates and are provided in counterclockwise order around the polygon,
- no adjacent edges of a tank are colinear,
- the corridor widths are unique,
- any pair of distinct corridor widths differ by at least 1 unit of distance, and
- corridor and tank model names in the input data are at most 15 characters in length.

Input

The input starts with the number of test cases on a line by itself. Each test case starts with the number of corridors, followed on the same line with a list of whitespace-separated width-and-name pairs, with the integer width and one-word names, separated by whitespace as well. The next line contains the number of tank models, following by that number of lines, one tank description per line. For each tank model, on its own line, the following is given: the number of vertices in the tank's footprint polygon, the name of the tank, and the x and y vertices of the poligon, in order around the polygon, with whitespace between all entries.

Output

For each tank model name provided in the input, produce one line of output, containing the name of the tank, followed by a space, followed by the narrowest corridor into which the tank will fit. The outputs for different test cases do not need to be separated in any special way.

(Continued on the next page.)

Sample Input

```
2
2
15 public_access 20 service_tunnel
5
4 square 0 0 18 0 18 18 0 18
3 triangle 1 1 10 -3 8 7
6 lumpy_diamond 0 14 14 0 21 6 28 14 21 22 14 28
4 smaller_diamond 0 14 14 0 28 14 14 28
4 tight_fit 1 1 16 1 16 16 1 16
1
10 back_door
1
3 triangle 1 1 100 -3 8 100
```

Sample Output

```
square service_tunnel
triangle public_access
lumpy_diamond none
smaller_diamond service_tunnel
tight_fit public_access
triangle none
```