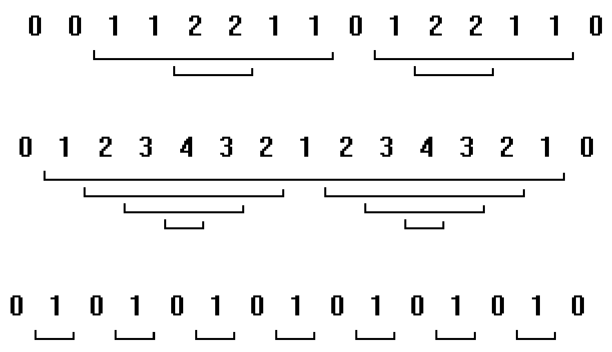
A • Islands in the Data Stream

Given a sequence of integers `a1, a2, a3, …, an`, an *island* in the sequence is a contiguous subsequence for which each element is greater than the elements immediately before and after the subsequence. In the examples below, each island in the sequence has a bracket below it. The bracket for an island contained within another island is below the bracket of the containing island.



Write a program that takes as input a sequence of **15** non-negative integers, in which each integer differs from the previous integer by at most **1**, and outputs the number of islands in the sequence.

Input

The first line of input contains a single integer ***P***, (**1** ≤ ***P*** ≤ **1000**), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, ***K,*** followed by **15** non-negative integers separated by a single space. The first and last integers in the sequence will be 0. Each integer will differ from the previous integer by at most 1.

Output

For each data set there is one line of output. The single output line consists of the data set number, ***K***, followed by a single space followed by the number of islands in the sequence.

Sample Input	Sample Output
4	1 4
1 0 0 1 1 2 2 1 1 0 1 2 2 1 1 0	2 7
2 0 1 2 3 4 3 2 1 2 3 4 3 2 1 0	3 7
3 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0	4 7
4 0 1 2 3 4 5 6 7 6 5 4 3 2 1 0	

B • Von Neumann’s Fly

The following problem was posed to John von Neumann:

> Two bicyclists, ***A*** and ***B***, start riding toward each other at the same time from places that are 250 miles apart, A traveling at 10 miles per hour, and ***B*** at 15 miles per hour. At the same time, a fly leaves the front wheel of ***A***'s bicycle, and flies toward ***B***'s bicycle at 20 miles per hour. As soon as he touches the front wheel of ***B***'s bicycle, he turns around and flies back. As the bicycles approach each other, he continues flying back and forth, touching each front wheel in turn, until, alas, he is crushed between them. Since the fly travels faster than either cyclist, he makes an infinite number of trips, yet travels a finite distance (the infinite series converges). How far did the fly travel?

Von Neumann immediately summed the infinite series (in his head!), and arrived at the correct answer: 200 miles.

You are to write a program that solves a more general version of that problem, with varying initial distances and speeds.

Input

The first line of input contains a single integer ***P***, (**1 ≤ *P* ≤ 1000**), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line containing five values: an integer ***N*** (the data set number), and four floating-point values: ***D*** (the initial distance between the bicycles, **10 ≤ *D* ≤ 1000**), ***A*** (cyclist ***A***'s speed in miles per hour, **1 ≤ *A* ≤ 30**), ***B*** (cyclist ***B***'s speed in miles per hour, **1 ≤ *B* ≤ 30**), and ***F*** (the fly's speed in miles per hour, ***A* ≤ *B*< *F* ≤ *50***).

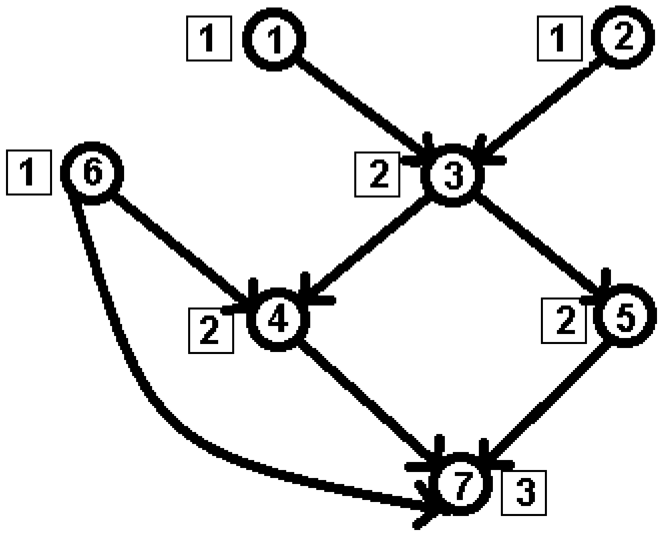
Output

For each data set there is one line of output. It contains the data set number followed by a single space, followed by the number of miles traveled by the fly, (the sum of the infinite series described by the input values), accurate to two decimal places.

Sample Input	**Sample Output**
`5`	`1 200.00`
`1 250 10 15 20`	`2 7.18`
`2 10.7 3.5 4.7 5.5`	`3 484.42`
`3 523.7 15.3 20.7 33.3`	`4 833.33`
`4 1000 30 30 50`	`5 416.67`
`5 500 15 15 25`	

C • Strahler Order

In geology, a river system can be represented as a directed graph. Each river segment is an edge; with the edge pointing the same way the water flows. Nodes are either the source of a river segment (for example, a lake or spring), where river segments merge or diverge, or the mouth of the river.



Note: The number in a box is the order. The number in a circle is the node number.

The *Strahler order* of a river system is computed as follows.

- The order of each source node is `1`.
- For every other node, let `i` be the highest order of all its upstream nodes. If just one upstream node has order `i`, then this node also has order `i`. If two or more upstream nodes have order `i`, then this node has order `i+1`.

The order of the entire river system is the order of the mouth node. In this problem, the river system will have just one mouth. In the picture above, the *Strahler* order is three (3).

You must write a program to determine the order of a given river system.

The actual river with the highest order is the *Amazon*, with order **12**. The highest in the U.S. is the *Mississippi*, with order **10**.

Node **M** is the mouth of the river. It has no outgoing edges.

Input

The first line of input contains a single integer ***K***, (**1 ≤ *K* ≤ 1000**), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of multiple lines of input. The first line of each data set contains three positive integers, ***K***, ***M*** and ***P*** (**2 ≤ *M* ≤ 1000**). ***K*** is the data set number. ***M*** is the number of nodes in the graph and ***P*** is the number of edges. The first line is followed by ***P*** lines, each describing an edge of the graph. The line will contain two positive integers, **A** and **B**, indicating that water flows from node ***A*** to node ***B*** (**1 ≤ *A*, *B* ≤ *M***). Node ***M*** is the mouth of the river. It has no outgoing edges.

Output

For each data set there is a single line of output. The line consists of the data set number, a single space and the order of the river system.

Sample Input	Sample Output

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

D • Pisano Periods

In 1960, Donald Wall of IBM, in White Plains, NY, proved that the series obtained by taking each element of the *Fibonacci* series modulo `m` was periodic.

For example, the first ten elements of the *Fibonacci* sequence, as well as their remainders modulo `11`, are:

```
n           | 1  2  3  4  5  6   7   8   9  10
F(n)        | 1  1  2  3  5  8  13  21  34  55
F(n) mod 11 | 1  1  2  3  5  8   2  10   1   0
```

The sequence made up of the remainders then repeats. Let `k(m)` be the length of the repeating subsequence; in this example, we see `k(11) = 10`.

Wall proved several other properties, some of which you may find interesting:

- If $m > 2$, $k(m)$ is even.
- For any even integer $n > 2$, there exists m such that $k(m) = n$.
- $k(m) \leq m^2 - 1$
- $k(2^n) = 3 * 2^{(n-1)}$
- $k(5^n) = 4 * 5^n$
- $k(2 * 5^n) = 6n$
- If $n > 2$, $k(10^n) = 15 * 10^{(n-1)}$

For this problem, you must write a program that calculates the length of the repeating subsequence, `k(m)`, for different modulo values `m`.

Input

The first line of input contains a single integer ***P***, (1 ≤ ***P*** ≤ 1000), which is the number of data sets that follow. Each data set is a single line that consists of two space separated integer values ***N*** and ***M***. ***N*** is the data set number. ***M*** is the modulo value (`2 <= m <= 1,000,000`).

Output

For each data set there is one line of output. It contains the data set number (***N***) followed by a single space, followed by the length of the repeating subsequence for ***M, k(M)***.

Sample Input	Sample Output
5	1 6
1 4	2 20
2 5	3 10
3 11	4 15456
4 123456	5 332808
5 987654	

E • Deranged Exams

The first question on the *Data Structures and Algorithms* final exam has a list of **N** terms and a second list of **N** definitions. Students are to match each term with the correct definition. Unfortunately, Joe, who wrote a Visual BASIC program in high school and assumed he knew all there was to know about Computer Science, did not bother to come to class or read the textbook. He has to guess randomly what the matches are. Let `S(N,k)` be the number of ways Joe can answer the question and get at least the first `k` matches wrong.

For this problem, you will write a program to compute `S(N,k)`.

Input

The first line of input contains a single integer ***P***, (**1** $\le$ ***P*** $\le$ **1000**), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input containing three space separated decimal integers. The first integer is the data set number. The second integer is the number, `N (1 <= N <= 17)`, of terms to be matched in the question. The third integer is the number, `k (0 <= k <= N)`, of initial matches to be incorrect.

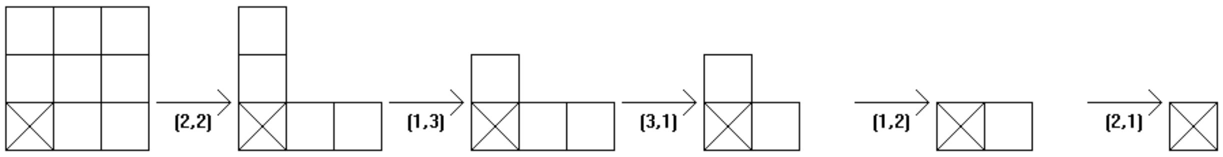
Output

For each data set there is a single line of output. It contains the data set number followed by a single space which is then followed by the value of `S(N,k)`.

Sample Input	Sample Output
4	1 18
1 4 1	2 3216
2 7 3	3 2170680
3 10 5	4 130850092279664
4 17 17	

F • Chomp

Chomp is a two-player strategy game played on a rectangular chocolate bar made up of smaller square blocks (cells). The players take turns choosing one block and "eating it" (removing it from the board), together with those that are above it and to its right. The bottom left block is *poisoned* and the player who is forced to *eat* it loses. The following diagram shows a game beginning with a 3-by-3 board. The **X** indicates the p*oisoned* cell.



A position in the game is a winning position if there is a move that results in a losing position for the opponent. A position in the game is a losing position, if every move from that position either *eats* the *poisoned* square (losing the game) or results in a winning position for the opponent.

In the example above, the 1x1 and equal armed **L**-shaped positions are losing positions (since the opponent can mirror the current player).

The 3x3, unequal armed **L** and 1x**n** positions are winning positions.

The aim of this problem is to “solve” 3-by-100 ***Chomp***. That is, for each possible position, determine whether it is a winning or losing position and if it is winning position, give the next move.

A position in 3-by-100 ***Chomp***, is determined by the number, ***p***, of squares in the bottom row, the number, ***q***, of squares in the middle row and the number, ***r***, of squares in the top row with:

$$100 >= p >= q >= r >= 0$$

Write a program which, for each possible position in the 3 by 100 game of ***Chomp***, determines whether it is a winning or losing position and if it is a winning position, gives the next move (the square to *eat* next).

Input

The first line of input contains a single integer P, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, ***K***, followed by the counts $100 >= p >= q >= r >= 0$ of squares in the bottom row (p), middle row (q) and top row ($\mathbf{r}$) respectively separated by single spaces.

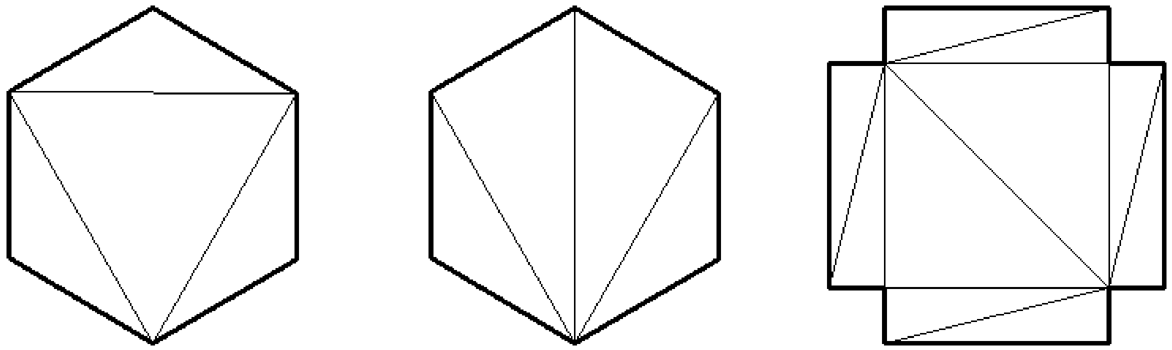
Output

For each data set there is a single line of output. If the input position is a losing position, the output line consists of the data set number, ***K***, followed by a single space followed by the (capital) letter **L**. Otherwise (the input position is a winning position), the output line consists of the data set number, ***K***, followed by the (capital) letter **W**, followed by the column number and row number of a block to *eat* which results in a losing position for the next player.

Sample Input	Sample Output
`4`	`1 W 2 2`
`1 3 3 3`	`2 W 3 1`
`2 3 1 0`	`3 L`
`3 3 2 0`	`4 W 51 1`
`4 97 64 35`	

G • Triangle Count Sequences of Polygon Triangulations

A *Triangulation* of a polygon is a set of lines between vertices of the polygon that divide the polygon into triangles. The lines must be inside the polygon and may not intersect each other except at vertices of the polygon. For example:



A **B** **C**

In the examples, the heavy line is the outline of the polygon and the lighter lines give the *triangulation*. In each case a triangulation of a polygon with ***n*** vertices will result in ***(n-2)*** triangles.

The *triangle count sequence* of a triangulation is obtained by starting at some vertex, proceeding around the polygon and for each vertex of the polygon recording the number of triangles of the triangulation incident on that vertex. In the examples above the triangle count sequences starting at the top (left) are:

```
A:  1 3 1 3 1 3

B:  2 2 1 4 1 2

C:  1 2 3 2 1 6 1 2 3 2 1 6
```

Write a program which takes as input a sequence of ***N*** positive integers and determines whether the sequence is the triangle count sequence of a polygon of ***N*** vertices. If so, list the triangles of the triangulation in lexicographical order.

Input

The first line of input contains a single integer ***P***, (**1** ≤ ***P*** ≤ **1000**), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, ***K,*** followed by the number, ***N***, (**4** <= ***N*** <= **20**) of integers in the sequence, followed by the ***N*** integers of the sequence in order, all separated by a single space.

Output

For each data set there may be multiple lines of output. If the input sequence is **NOT** the *triangle count sequence* of a polygon of ***N*** vertices, the single output line consists of the data set number, ***K***, followed by a single space followed by the (capital) letter **N**. Otherwise (the input sequence is the *triangle count sequence* of a polygon of ***N*** vertices), the first output line consists of the data set number, ***K***, followed by a single space followed by the (capital) letter **Y**. Following this line are (**N-2**) lines each containing the indices of the vertices of a triangle of the triangulation, one triangle per line, with the vertex indices in increasing order. The triangles should be sorted in *lexicographic* order least first. That is, if **k** < **l** < **m** are the vertex indices of one triangle and **r** < **s** < **t** are the vertex indices of another, then the first precedes the second in *lexicographical* order if:

k < **r** OR (**k** == **r** AND **l** < **s**) OR (**k** == **r** AND **l** == **s** AND **m** < **t**)

Sample Input	Sample Output

4
1 6 1 3 1 3 1 3
2 6 1 3 2 2 1 3
3 12 1 2 3 2 1 6 1 2 3 2 1 6
4 20 1 2 3 2 1 6 1 2 3 2 1 6 1 3 2 2 1 3 1 2
	1 Y
	1 2 6
	2 3 4
	2 4 6
	4 5 6
	2 N
	3 Y
	1 2 12
	2 3 12
	3 4 6
	3 6 12
	4 5 6
	6 7 8
	6 8 9
	6 9 12
	9 10 12
	10 11 12
	4 N

(Note: The first line of output for each input data set is shown in **bold** to make it easier to read.)

H • Powers of Pascal

The *Pascal matrix* is the (infinite) matrix defined by (zero based row and column):

```
Pascal[row, column] = Comb(row, column) for 0 <= column <= row
```

and zero otherwise, where **`Comb(n, k)`** is the number of combinations of **`n`** things taken **`k`** at a time (the binomial coefficient).

```
1    0    0    0    0    0    0    0    0    0 …
1    1    0    0    0    0    0    0    0    0 …
1    2    1    0    0    0    0    0    0    0 …
1    3    3    1    0    0    0    0    0    0 …
1    4    6    4    1    0    0    0    0    0 …
1    5   10   10    5    1    0    0    0    0 …
1    6   15   20   15    6    1    0    0    0 …
1    7   21   35   35   21    7    1    0    0 …
1    8   28   56   70   56   28    8    1    0 …
1    9   36   84  126  126   84   36    9    1 …
.    .    .    .    .    .    .    .    .    .
.    .    .    .    .    .    .    .    .    .
.    .    .    .    .    .    .    .    .    .
```

For this problem, you will write a program to compute entries in powers of the *Pascal* matrix:

$$\texttt{Pascal}^{P} = \texttt{Pascal} \times \texttt{Pascal} \times \ldots \times \texttt{Pascal} \quad (\textbf{P} \text{ factors})$$

Since the matrix is lower triangular, all powers are lower triangular and only the upper left **`N`** by **`N`** corner is used in computing coefficients in the upper left **`N`** by **`N`** corner of the power.

Input

The first line of input contains a single integer *K*, (1 ≤ *K* ≤ 1000), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input containing four space-separated decimal integers. The first integer is the data set number. The second integer is the power, **`P`** `(`**`1 <= P <= 100,000`**`)`, to which to raise the Pascal matrix. The third and fourth integers give the row number, **`R`**, and the column number, **`C`**, of the desired entry `(`**`0 <= C <= R <= 100,000`**`)`.

Output

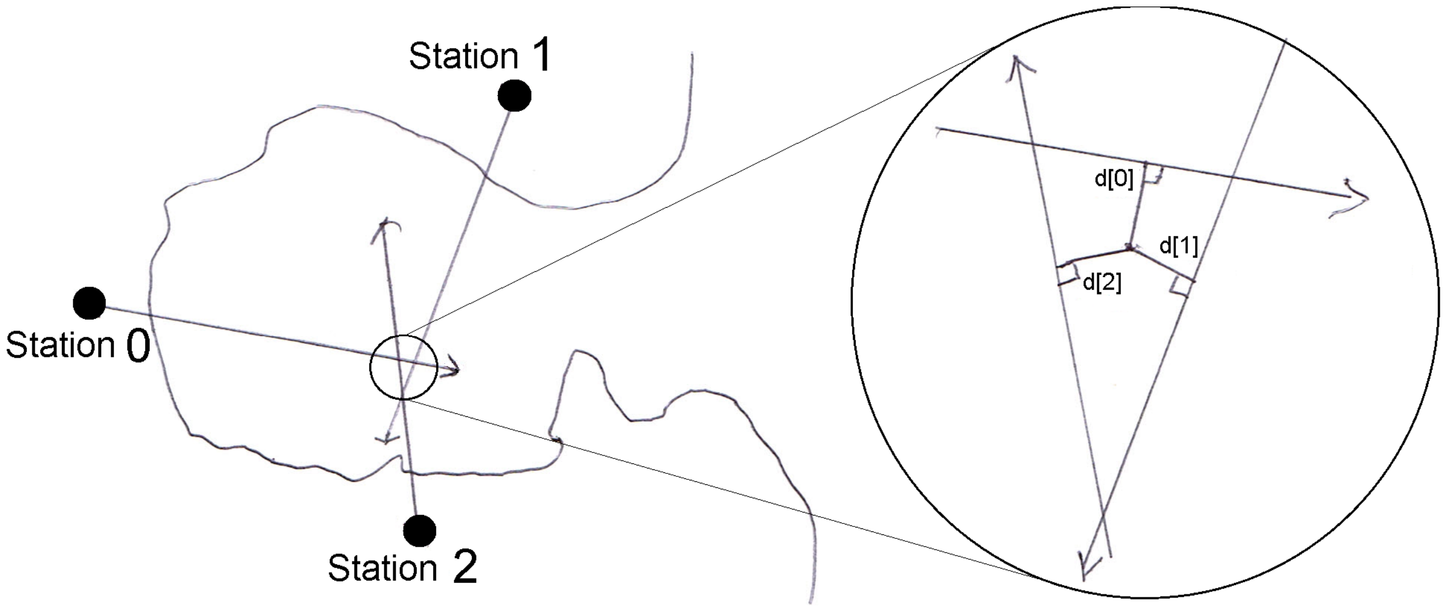
For each data set there is a single line of output. The line consists of the data set number, a single space, which is then followed by the requested entry of the requested *Powers of the Pascal* matrix. Input values will be restricted so results will not overflow a 64-bit integer value.

Sample Input	Sample Output

3
1 1 8 3
2 9 21 13
3 200 100000 99998
	1 56
	2 8759577256290
	3 199998000000000

I • Contraband

It has been reported that something fell out of an aircraft approaching the airport over the bay. Thinking the object may have been some sort of contraband to be picked up by a confederate, the police want to watch for a repeat whenever any of the aircraft that could have been the one that dropped the object again approaches the airport over the bay. Three observation stations with night vision equipment have been stationed around the bay (see figure below).



Intersection Detail

Station 1 is 3.715 kilometers east and 1.765 kilometers north of station 0 and station 2 is 2.894 kilometers east and 2.115 kilometers south of station 0.

When a suspect aircraft crosses the bay, each observer follows it with the night vision equipment while in contact with the others. If any observer sees something falling from the aircraft, each records a direction to the object and a *confidence level* for that direction. The *confidence level* (**CL**) is a value from 0 (more or less pointing at the aircraft) to 1 (pointing at the splash where the object hit the water).

In general, the three sight lines will not cross at a single point but will form a triangle (see *Intersection Detail* above). The best estimate of the actual position is to be the point **(x, y)** which minimizes the sum of the squares of the distances, **d[i]**, to each line, weighted by the confidence level, **CL[i] + 0.2**,

$$\text{Minimize SUM}(i = 0 \text{ to } 2)\ \{(CL[i] + 0.2) * d[i]^2\}$$

For this problem, you will write a program which takes as input the three observer directions and the three confidence levels and outputs the point **(x, y)**, which minimizes the above sum, where **x** is the distance in kilometers east of station 0 and y is the distance in kilometers north (positive) or south (negative) of station 0.

Input

The first line of input contains a single integer ***P***, (1 ≤ ***P*** ≤ 1000), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set is a single line of input consisting of the data set number ***N***, followed by a space, followed by six space separated floating point values. The floating point values are, in order, **a[0]**, **CL[0]**, **a[1]**, **CL[1]**, **a[2]**, **CL[2]**. **a[i]** is the bearing (in degrees clockwise from north) from station **i** (**0 ≤ a[i] < 360**), and **CL[i]** is the confidence level of observer **i** (**0 ≤ CL[i] ≤ 1**).

Output

For each data set there is one line of output. It contains the data set number, ***N***, followed by a single space which is then followed by two space separated values, **x** and **y**. **x** is the distance east of station 0 in kilometers, and **y** is the distance north (positive) or south (negative) of station 0. The distances should be displayed to 3 decimal places.

Sample Input	Sample Output

3
1 44.0 0.38 272.9 0.41 345.5 0.64
2 43.5 0.80 263.6 0.81 338.2 0.83
3 45.9 0.50 279.2 0.78 348.7 0.81
	1 1.847 1.877
	2 1.440 1.511
	3 2.073 2.021

Problem J
Fast Food Prizes

Around regional contest time, the Canadian branch of a popular fast food restaurant usually runs a game to promote its business. Certain food items provide stickers, and certain collection of different stickers can be converted to cash prizes. If a prize requires sticker types $T_1, T_2, \ldots, T_k$, then you can claim the prize if you have 1 sticker of each type $T_1, T_2, \ldots, T_k$. Each sticker can only be used to claim one prize. However, you may claim a prize multiple times if you have multiple stickers of the same type. No two prizes will require the same type of stickers. There may be some stickers that cannot be used to claim a cash prize (e.g. a sticker for a free milkshake).

On your road trip to the regional contest, your coach forced you to eat at this restaurant and collected all the stickers together. How much cash can your coach claim?

Input

The input consists of multiple test cases. The first line of input is a single integer, not more than 1000, indicating the number of test cases to follow. Each case starts with a line containing two integers n ($1 \leq n \leq 10$) and m ($1 \leq m \leq 30$), where n is the number of different types of prizes, and m is the number of different types of stickers (the types are labelled $1, 2, \ldots, m$). The next n lines specify the prizes. Each of these lines starts with an integer k ($1 \leq k \leq m$) specifying the number of sticker types required to claim the prize. This is followed by k integers specifying the types of the stickers required. The final integer on each line is the (positive) cash value of the prize (at most 1,000,000). The last line of each case gives m nonnegative integers, with the ith integer giving the number of stickers of type i your coach has collected. There are no more than 100 stickers of each type.

Output

For each case, display on a single line the total value of the cash prizes that can be claimed.

Sample Input

```
3
2 10
3 1 2 3 100
4 4 5 6 7 200
2 3 1 4 5 2 2 1 3 4
3 6
2 1 2 100
3 3 4 5 200
1 6 300
1 2 3 4 5 6
3 6
2 1 2 100
3 3 4 5 200
1 6 300
1 2 0 4 5 6
```

Sample Output

```
500
2500
1900
```

Problem K
Social Advertising

You have decided to start up a new social networking company. Other existing popular social networks already have billions of users, so the only way to compete with them is to include novel features no other networks have.

Your company has decided to market to advertisers a cheaper way to charge for advertisements (ads). The advertiser chooses which users' "wall" the ads would appear on, and only those ads are charged. When an ad is posted on a user's wall, all of his/her friends (and of course the user himself/herself) will see the ad. In this way, an advertiser only has to pay for a small number of ads to reach many more users.

You would like to post ads to a particular group of users with the minimum cost. You already have the "friends list" of each of these users, and you want to determine the smallest number of ads you have to post in order to reach every user in this group. In this social network, if A is a friend of B, then B is also a friend of A for any two users A and B.

Input

The input consists of multiple test cases. The first line of input is a single integer, not more than 10, indicating the number of test cases to follow. Each case starts with a line containing an integer n ($1 \leq n \leq 20$) indicating the number of users in the group. For the next n lines, the ith line contains the friend list of user i (users are labelled $1, \ldots, n$). Each line starts with an integer d ($0 \leq d < n$) followed by d labels of the friends. No user is a friend of himself/herself.

Output

For each case, display on a line the minimum number of ads needed to be placed in order for them to reach the entire group of users.

Sample Input

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

Sample Output

```
1
2
```

Problem L
Tables

HTML uses a simple tag format for table layout. You are to create ASCII-art tables based on a simplified notation.

Logically, a table can be considered an m by n grid, each with a 2-character wide by 1-character high cell, for example a 2×3 grid would be as follows:

```
 -- -- --
|11|12|13|
 -- -- --
|21|22|23|
 -- -- --
```

The output consists of $2m + 1$ rows, each having $3n + 1$ characters (including leading and trailing spaces in odd-numbered rows).

But some tables are not strictly grid-based, because the values for certain cells can span multiple rows and/or columns. Here is the layout if cell 11 has a row span of 2 and cell 22 has a column span of 2:

```
 -- -- --
|11|12|13|
    -- --
|  |22   |
 -- -- --
```

You are to create these ASCII-art tables given m, and the row span and column span values.

Input

The input consists of multiple test cases. The first line of each test case contains one integer m, the number of rows. Then, m lines follow, describing the row and column spans in the layout. Each of the next m lines gives the span information for a row. The span information is given only for the upper left cell in the span. If N row spans and column spans need to be specified for a given row, these would be specified as:

$$N\ \texttt{RS}_1\ \texttt{CS}_1\ \ldots\ \texttt{RS}_N\ \texttt{CS}_N$$

where the values $\texttt{RS}_k$ and $\texttt{CS}_k$ are between 1 and 9, and specify how many rows and columns the next cell needing information in this row needs to occupy. Previous row spans or column spans may imply that fewer cells than the total number of columns need to be specified for a particular row.

You may assume that the input is valid:

- the number of rows and columns are between 1 and 9;
- every row will have the same number of columns;

- there are no overlapping spans of cells;
- every cell is contained in some span of cells.

There are at most 100 test cases and the end of input is indicated by $m = 0$.

Output

For each case, display the ASCII-art table, with the row-column index displayed in the top left corner of every span of cells. Display a blank line after every case.

Sample Input

```
3
2 1 1 1 1
2 1 1 1 1
2 1 1 1 1
3
2 1 1 2 1
1 1 1
2 1 1 1 1
3
1 1 2
2 1 1 1 1
2 1 1 1 1
3
2 2 1 1 2
1 2 2
1 1 1
0
```

Sample Output

```
 -- --
|11|12|
 -- --
|21|22|
 -- --
|31|32|
 -- --

 -- --
|11|12|
 --
|21|  |
 -- --
|31|32|
 -- --

 -- --
|11   |
 -- --
|21|22|
 -- --
|31|32|
 -- --

 -- -- --
|11|12   |
    -- --
|  |22   |
 --
|31|     |
 -- -- --
```