**2014 ACM-ICPC Regional Programming Contest**

**DEPARTMENT OF COMPUTER ENGINEERING**

**FACULTY OF ENGINEERING - CHULALONGKORN UNIVERSITY**

**November 15, 2014**

**Contest Session**

**Task Setters**

Apirak Hoonlor

Nattee Niparnan

Nuttapong Chenthanez

Pichayoot Ouppaphan

Pinyo Taeprasartsit

Sanpawat Kantabutra

Shahriar Manzoor

Mahidol University, Thailand

Chulalongkorn University, Thailand

Chulalongkorn University, Thailand

Mahidol Wittayanusorn, Thailand

Silpakorn University, Thailand

Chiang Mai University, Thailand

Southeast University, Bangladesh

**Special Thanks**

Derek Kisman, Mana Borwornphadungkitti, Mohammad Hedayetul Islam, Mohammad Mahbubul Hasan, Nasakol Pongkorpsakol, Nutdanai Phansooksai, Park Netrakom, Pattara Sukprasert, Rujia Liu, Sukhum Sattaratnamai, Tanaphon Tangngamjit , Thanoos Keadlarb, Watcharapol Watcharawisetkul, Wongsagorn Yothinwat



**Time limit for each problem (second)**

|  |  |
| --- | --- |
| **A** | **7** |
| **B** | **2** |
| **C** | **3** |
| **D** | **2** |
| **E** | **1** |
| **F** | **1** |
| **G** | **2** |
| **H** | **1** |
| **I** | **1** |
| **J** | **1** |
| **K** | **1** |
| **L** | **1** |

**Memory limit : 512 MB (All problems)**

|  |  |
| --- | --- |
| **A** | **Volume Control** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

Controlling the volume level is important for computers, smart phones or any hardware that plays music or songs or acts as a sound mixing device. Most such devices have one master volume controller and one or more secondary volume controller. Having more than one controller is useful for two reasons: (i) We can keep volume levels different for different programs (Software)/ devices (Hardware) (ii) Higher precision volume control.

The first reason is common and well known to all of us but in this problem we will concentrate on the second reason. In the figure on the left, a volume control interface is shown. Using the master volume controller one can set the general volume level for all sound/mixing devices and using the other volume controllers, one can set volume level for individual sound devices/software. For example after midnight one would like to keep the headphone sound level high but the Speaker sound level low for obvious reasons. But for high quality sound devices this two level controlling can ensure higher precision volume controlling. For example if the master volume controller can set 100 different discrete volume levels (Equidistant levels of course) it cannot actually set more than 100 levels, but if the Speaker volume controller can also set 100 different discrete volume levels then in combination they can set a lot more than 100 volume levels. If both the master and the speaker volume controller is at 50% level, in actually the speaker is at (0.500.50100=) 25% volume level (Considering 100% as the highest possible volume level), if the master and the speaker volume controller is at 30% and 42% level respectively then the speaker is actually at level 12.6 %. Given the number of different volume levels both the Master and the secondary volume controller can set you have to report how many different volume levels they can create combined.

**Input**

First line of the input file contains a positive integer **T** (**T** ≤ 10,000) which indicates the total number of test cases. Each of the next line contains a positive integer **N** (**N** ≤ 30,000) which denotes how many different volume levels (Excluding the 0% level) both the Master and secondary volume controller can set.

**Output**

For each test case produce one line of output. This line contains an integer **D** which denotes how many different volume levels can be set using the Master and secondary volume controller.

**Note**

Illustration of the 2nd sample input. The master volume controller can set volumes to 0%, 25%, 50%, 75% and 100% and the secondary volume controller can reduce this again to 0%, 25%, 50%, 75% and 100%. So when these two are combined they can set volume levels to 0%, 0%, 0%, 0%, 0%, 0%, 6.25%, 12.5%, 18.75%, 25%, 0%, 12.5%, 25%, 37.5%, 50%, 0%, 18.75%, 37.5%, 56.25%, 75%, 0%, 25%, 50%, 75%, 100%. So unique volume levels these two can set are 0%, 6.25%, 12.5%, 18.75%, 25%, 37.5%, 50%, 56.25%, 75% and 100%, so total 10 different volume levels.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 234  | 710 |

|  |  |
| --- | --- |
| **B** | **Combination** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

The number of ways in which **r** objects can be chosen from **n** different objects can be found using the formula. For example  etc. Now if **n** varies from **low** to **high** and **r** varies from **0** to **n**, then you have to find out how many values of  are odd. In other words you will have to find out the value of , here **mod** is the standard modulus or reminder operation.

**Input**

The input file contains at most 50,000 lines of inputs. Each line contains two positive integers **low** and **high** (0 ≤ **low** ≤ **high** ≤ 16\*1011). Input is terminated by a line containing two zeroes.

**Output**

For each line of input, produce one line of output. This line contains an integer **D** which prints the desired value. You can safely assume that this output will fit in a 64-bit unsigned integer.

**Note**

Illustration for Sample input 1:  , and of these seven values, six (6) are odd.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 310 20100 2000 0 | 6702510 |

|  |  |
| --- | --- |
| **C** | **Mesh Cutter** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

Alan is building a small 3D modeling program that aims to be extremely light-weight yet powerful enough to solve a large number of practical problems. However, he is stuck with the “knife” operation: given a mesh, a plane and a point, the knife operation cuts the mesh with the plane, and removes the part that is in the same half-space as the point.

 Knowing that programs written in ACM/ICPC competitions are usually very compact, Alan comes for your help. In this problem, you only need to deal with “nice” solid *polygonal* meshes (3D experts told you that pure *triangular* meshes are too restrictive for editing), like this:



To be precise, “nice” means:

1. There will be no duplicated vertices/edges/faces.
2. Faces are planar convex polygons, **usually triangles or quads**.
3. The mesh is an **orientable manifold without boundary**.
4. The faces enclose a non-empty connected part of space (so we say it’s “solid”), and there will be no hidden faces (i.e. no faces are invisible from outside).

For those of you who are unfamiliar with terms in point 3, it means:

1. Every edge is incident to **exactly** two faces. So when you walk across an edge from a face, you will not directly reach the back side of that face (hence “no boundary”).
2. The faces incident to a vertex form a **closed fan**, see below. Note that if the faces form an open fan, the boundary edges are violating point 7 because they are incident to only one face.



1. It’s not something like the famous Mobius band.



Note that despite the nice properties above, those “real-world” meshes are still not easy to deal with:

1. Two adjacent edges on a face can be **collinear** (but not overlapping).
2. Two adjacent faces (i.e. share a common edge) **can be co-planar** (but not overlapping).
3. The order of vertices in each face is either clockwise or counter-clockwise. That means the surface normal either point towards or away from the solid.

Your task is to compute the volume, surface area of the meshes **after cut**, as well as the shape of cross-section.

**Input**

The input will contain at most 25 test cases. Each test case begins with two integers **n**, **f** (4 ≤ **n** ≤ 1,000; 4 ≤ **f** ≤ 1,000), the number of vertices and faces. Each of the following n lines contains three real numbers **x**, **y**, **z**, the coordinates of the vertices. Each of the following f lines describes a face. Each line contains a sequence of integers beginning with **v** (3 ≤ **v** ≤ 10), the number of vertices in the face (vertices are numbered 1 to n), followed by a sequence of vertices in the face. The faces are guaranteed to form a single connected solid.

The final line of each test case contains 12 real numbers: **x1**, **y1**, **z1**, **x2**, **y2**, **z2**, **x3**, **y3**, **z3**, **x4**, **y4**, **z4** that means the solid will be cut with the plane containing triangle P1(x1, y1, z1) - P2(x2, y2, z2) - P3 (x3, y3, z3), and you need to remove the half-space containing point P4(x4, y4, z4). It is guaranteed that P1P2P3 is a valid triangle, and P4 is not on the plane of P1P2P3. Coordinates have absolute values not greater than 100.

 **Important:** coordinates will be either exact (like integers or finite real numbers like 0.5) or given with enough precision, so you don’t have to worry about precision problems like seemingly non-planar faces. Furthermore, the distance between any mesh vertex and the cut-plane is at least 0.01, so don’t worry about degenerated cases.

**Output**

For each test case, print the case number and 5 lines. The first 3 lines are about the solids after cut.

**Line 1:** The number of connected solids after cut.

**Line 2:** The volumes of these solids, in decreasing order.

**Line 3:** The surface areas of these solids, in decreasing order.

It is guaranteed that the resulting solids will not touch each other. If nothing is left after cut, line 2 and line 3 should be empty.

Next two lines are about the cross-section.

**Line 4:** The number of connected regions in the cross-section.

**Line 5:** The areas of these connected regions, in decreasing order.

Note that adjacent co-planar faces in the cross-section should be considered as in the same connected region. Beware that they may contain holes (and the holes can also enclose other connected regions). If the cross-section is empty, line 5 should be empty.

All values representing volumes or areas should be rounded to 3 decimal places, and an absolute error of up to 10-3 is allowed. It is guaranteed that all these volumes and areas are greater than 10-2.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 8 60 0 01 0 01 1 00 1 00 0 11 0 11 1 10 1 14 1 4 3 24 5 6 7 84 1 2 6 54 2 3 7 64 3 4 8 74 4 1 5 80 0 0.5 1 0 0.5 1 1 0.5 0 0 18 80 0 05 3 00 1 0-4 3 00 0 15 3 10 1 1-4 3 13 1 2 33 3 4 13 5 6 73 7 8 54 1 2 6 54 2 3 7 64 3 4 8 74 4 1 5 8-10 2 0 10 2 0 -10 2 1 0 0 0 | Case 1:10.5004.00011.000Case 2:20.417 0.3336.303 5.23620.833 0.667 |

**Explanation**

The second case is:



**Note**

Be sure that your program can handle complex meshes like this one (one of judge test case):

****

|  |  |
| --- | --- |
| **D** | **Optimal Landing Location** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |



You own a large island where you want to build an industry of your own. But the water around the island is not very deep so you can only use small boats to carry goods, but over the years this has proven to be a very slow process and also a risky one. So now you have bought a large Cargo Plane which can land anywhere on the island. Your job is to find an optimal place for landing. Let us denote is optimal place as L. Some additional information to solve this problem are:

* You already have three warehouses in the island to store goods.
* You know the how many goods can be kept in each warehouse and the amount of goods in the cargo plane. These are denoted by four positive integers.
* From the landing location L you plan to use three different trucks to carry cargo to the three warehouses. Each truck will start from L and reach the desired warehouse using the straight-line route, and a truck will not unload in more than one warehouse under any circumstances. After unloading, a truck will return to L using the same route.
* Multiple trips can be used to take all the goods to the warehouses. But after delivering all goods to the warehouses the three trucks will finally return to L.
* If the total amount of goods you can keep in the three warehouses is greater than goods bought via plane, you can decide what amount to be put in which warehouse.
* The capacity of all three trucks is 1. So each of the truck can take one unit of goods in each trip.

Given the location of the three warehouses, their capacity and the amount of goods bought by the Cargo Plane, your job is to find such a location for landing so that the total distance covered by three trucks is minimum. You can assume that when the plane arrives all three warehouses are empty. You don’t have to print the location of the landing location but only need to print the minimum total distance covered by the three trucks. Note that you have been asked to minimize the total distance covered by three trucks and need not to minimize the total time needed to carry the goods to the warehouses.

**Input**

First line of the input file contains a positive integer **N** (**N** ≤ 20,000) which denotes the number of test cases. The input for each test case is given in two lines and the description of these two lines is given below:

First line of each test contains six integers **Ax**, **Ay**, **Bx**, **By**, **Cx** and **Cy**
(0 ≤ **Ax**, **Ay**, **Bx**, **By**, **Cx**, **Cy** ≤ 1,000), these integers denote that (Ax, Ay), (Bx, By) and (Cx, Cy) are the locations of three warehouses A, B and C respectively in Cartesian coordinate system. You can assume that these points are not collinear. The second line contains four integers **CA**, **CB**, **CC** and **WP** (20 ≤ 2**CA**, 2**CB**, 2**CC**, **WP** ≤ 2,000 and **WP** ≤ **CA**+ **CB**+ **CC**) which denotes the capacity of the three warehouses A, B, C and the amount of goods bought by the plane respectively.

**Output**

For each set of input produce one line of output. This line contains a floating point number which denotes total distance covered by the three trucks when the cargo plane lands at an optimal location. This number should have four digits after the decimal point. An optimal location is the location from which the cost (sum of total distance covered by the three trucks) of carrying the goods to the warehouses is minimum.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 110 10 100 10 50 8010 10 10 30 | Case 1: 2960.5351 |

|  |  |
| --- | --- |
| **E** | **Zeroes** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

Factorial n is written as n! and n!=1\*2\*3...\*(n-1)\*n. For example 2! = 1\*2 = 2, 3! = 1\*2\*3 = 6, 5! = 120, 10! = 3,628,800 etc. The function fzero(n) denotes the number of trailing zeroes in n! in decimal number system. For example fzero(2) = 0, fzero(5) = 1, fzero(10) = 2. Given the domain of the input parameter v of fzero(v) function, you will have to find out how many different values of fzero() are there within this range.

**Input**

The input file contains at most 50001 lines of inputs. Each line contains two positive integers **low** and **high** (0 < **low** ≤ **high** ≤ 9\*1018). Input is terminated by a line containing two zeroes.

**Output**

For each line of input produce one line of output. This line contains an integer D, which denotes how many different values the function fzero(v) can have if (low ≤ v ≤ high).

**Note**

Illustration for Sample input 1: as 1! = 1, 2! = 2, 3! = 6, 4! = 24, 5! = 120, 6! = 720,
7! = 5,040, 8! = 40,320, 9! = 362,880, 10! = 3,628,800, so fzero(1) = 0, fzero(2) = 0,
fzero(3) = 0, fzero(4) = 0, fzero(5) = 1, fzero(6) = 1, fzero(7) = 1, fzero(8) = 1, fzero(9) = 1 and fzero(10) = 2. So in this range (1 to 10) there are three different values of fzero(v) => 0, 1 and 2.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 101 30 0 | 31 |

|  |  |
| --- | --- |
| **F** | **Fishing** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

You are a captain of a large fishing ship and you are on a fishing mission in the ocean. The ocean can be modeled by a grid of **R** rows and **C** columns. Each cell in the grid can be identified by a pair (r, c) which denotes the row and the column of the cell. In the cell (a, b), there are D[a][b] units of fish. Obviously, we want to fish as many fishes as possible. However, overfishing is bad for the environment. This ocean has a rule to prevent overfishing.

The rules state that for every cell that you fish, you have to nourish at least two other cells. To nourish a cell is to simply provide food for the fish in the cell. Additionally, there are two more rules. First, if the latest cell that we fish is at cell (a, b), the next cell that we fish must be at cell (p, q) where p > a and q > b. Second, if the latest cell that we nourished is at cell (a, b) the next cell that we nourish must be at cell (p, q) where p > a and q > b. These rules force us to not overfishing at particular cell and to distribute nourishing.

Your task is to calculate maximum possible profit from this fishing. Assuming that the cost to nourish one unit of fish is equal to the profit from one unit of fish, the lump sum benefit of the mission is the summation of cells that we fish minus the summation of the cells that we nourish. Finally, you have to start by nourishing at least 2 cells. In the worst case, we can choose to do nothing at all, resulting in zero benefit.

Be noted that fishing and nourishing do not change the number of fishes in the cell. When fishing or nourishing in any cell, we have to fish or nourish all the fishes in that cell.

The following list of cells are a possible route of our ship: (0,0) → (2, 3) →(1, 1) → (3,5) → (4,7) → (4,7), where the underlined cell is the cell that we fish. Be noted that it is possible to fish at the cell that we just nourish. The only rule for fishing at cell (p, q) is that the previous cell that we fish is at (a, b) where p > a and q > b. The cell that we nourish does not limit the cell that we fish. Similarly, the only rule for nourishing at cell (p, q) is that the previous nourishing cell is at (a, b) where p > a and q > b. The cell that we fish does not limit the cell that we nourish.

**Input**

There are multiple test cases. The first line of input contains an integer **T** (**T** ≤ 10) that indicates the number of test cases. Then T test cases follow, each uses the following format.

* The first line contains two integer **R** and **C** (1 ≤ **R**, **C** ≤ 100), the number of rows and columns of the grid.
* The next R lines describe the number of fishes in the grid. Each line describes one row of the grid, starting from row 0 to row R-1. There are C integers in each lines, each describing the value of the cell in that particular row, starting from column 0 to column C - 1. The number of fishes in each cell is non-negative integer not exceeding 1,000.

**Output**

For each test case, display one line of output containing the maximum possible benefit that we can get.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 24 41 1 1 41 3 1 11 1 2 11 1 1 13 51 1 1 1 11 1 1 1 11 1 1 1 1 | 20 |

|  |  |
| --- | --- |
| **G** | **Landmine Cleaner** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

In this problem, you are in a Landmine Cleaner (LC) team. Landmine Cleaner’s job is to identify and dispose of all landmines in a field so that troops can safely march through the field. Obviously, this is an extremely dangerous job! The first step in the process of landmine disposal is to identify where they are. Luckily, you have a Landmine Detector Drone (LDD) to help you locate the mines. LDD works by flying over a suspected area and send out a Z-ray, which is a military classified ray. If there is a mine below, the Z-ray will reflect from the mine and the LDD will pick-up a strong signal. LDD however, is not perfect, because nearby mines can also reflect Z-ray and cause LDD to pick-up some amount of signal, albeit weaker. Your job in this problem is to determine where the mines are using the information from LDD signals. Remember, your fellow teammates’ life depend critically on the correctness of your solution!

The field in this problem is an **N** (1 ≤ **N** ≤ 1,000) rows by **M** (1 ≤ **M** ≤ 1,000) columns grid of cells. A cell can either have 1 or 0 mine at the center. You are given the signal reading from LDD when it flies over the center of each cell. Consider 3x3 cells whose center cell is where your LDD is at. The LDD signal reading at the center cell will be equal to:

If the center cell has landmine, 3 + number of landmines in the 3x3 cells

If the center cell does not have landmine, 0 + number of landmines in the 3x3 cells

In the following examples, L indicates that there is a landmine in that cell, while - indicates that there is no landmine in that cell.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| L | - | - |  | - | - | - |  | L | L | - |  | L | L | L |  | - | L | L |  | L | L | - |
| - | L | - |  | - | - | - |  | - | L | - |  | L | L | L |  | L | - | L |  | L | - | - |
| - | - | - |  | - | - | - |  | - | L | - |  | L | L | L |  | - | L | L |  | L | L | - |

LDD at center cell: 5 0 7 12 6 5

Your task in this problem is to determine for each cell if there is a landmine or not.

**Note**

You can safely assume that there is no landmine outside the field. You can also be sure that the solution is unique.

**Input**

There are **T** test cases. The first line contains the number **T** (1 ≤ **T** ≤ 10). Then, T test cases follow, each using the following format.

* The first line consists of two integers **N** and **M**.
* Each of the next N line has M integers specify the LDD signal reading at each cell, each row per line, starting from the first row to the last row. In each line, there are M integers which indicate the reading of the specific row, starting from the first column to the last column.

**Output**

For each test case, you have to output N lines, each line is a length M string of ‘-’ and ‘L’ indicating whether there is a landmine or not.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 13 42 6 3 27 5 7 56 7 3 2 | -L--L-LLLL-- |

|  |  |
| --- | --- |
| **H** | **Hidden Plus Signs** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

In this problem, we want to find plus signs hidden in a number table. All we know about these plus signs and the number table are the followings.

1. The plus signs may overlap.
2. The width and height of each plus sign range from 3 to 11 units. For each plus sign, its width and height are equal.
3. One unit length equals to one table entry.
4. There are 2 to 9 plus signs in each table and no part of a plus sign lies outside a table.
5. The centers of plus signs do not lie inside another plus sign. In other words, the center of each sign does not overlap with any part of another plus sign.
6. The value of each table entry is the number of plus signs in the entry. For example, if there are 3 plus signs overlap in a table entry, the value in this entry is 3. If there is none, the value is 0.

For example, there are 2 plus signs in the following table. One sign has width of length 3 and another has width of length 5. The center of the first lies at row 2, column 2 of the number table, while the center of the second lies in row 3, column 3. The highlighted entries correspond to the centers of plus signs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 2 | 0 | 0 |
| 1 | 2 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |

Your task is to write a program to count the number of plus signs in each table and report the position of the last plus sign's center. By last, it means the plus sign whose center lies lowest in a table. If there are multiple signs lie lowest, the center of the last plus sign is the rightmost among them.

**Input**

The first is the number of test cases **T** where 1 ≤ **T** ≤ 12. For each test case:

1. The first line is the number of rows **R** and the number of columns **C** of the table where 3 ≤ **R**, **C** ≤ 30
2. The next R lines are the value of table entries from the top to bottom rows.

**Output**

For each test case, your program will write two output lines. The first is the number of plus signs hidden in a table. The second is the row and column of the last plus sign's center.

**Example**

|  |  |
| --- | --- |
| Input | Output |
| 25 50 1 1 0 01 1 2 0 01 2 1 1 10 0 1 0 00 0 1 0 010 110 0 0 0 1 1 0 0 0 0 00 0 0 0 1 1 0 1 0 0 00 0 1 1 1 2 2 1 1 0 00 0 1 2 2 1 1 2 2 0 00 0 1 1 3 1 0 0 1 0 00 0 0 2 1 2 2 1 1 1 10 1 0 0 1 1 1 0 1 1 01 1 1 0 1 1 1 1 3 1 10 1 0 0 0 0 1 0 0 1 00 0 0 0 0 0 1 0 0 0 0 | 23 398 10 |

**Note**

Hint for Example Test Case 2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 0 | 0 |
| 0 | 0 | 1 | 1 | 3 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

|  |  |
| --- | --- |
| **I** | **The Programmers** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

The Programmers is a popular reality show focusing on programming contest. Each year many people would enter the contest, and compete to be the next top programmer. Due to its popularity, several people would like to enter the show. The Programmers organizes local contests around Thailand to find the great talents. Each local contest site can only handle **C** contestants. However, due to exhausted problem setters, there are only **S** local-contest sites, numbered from 1 to S. Note that all local contests are organized at the same time. Hence, one contestant can only participate in at most one of the local contest.

The local contest sites are located all over the country. For some contestants, it is not possible to go to the sites that are very far from their home town. To facilitate the contestants, the organizer asks each contestant to list local contest sites that they can join.

Some contestant might not be able to compete because these constrains. Your task is to calculate the maximum overall number of contestants that can participate in the local contests without breaking these constrains.

**Input**

In the first line of input contains a single integer, **n** – the number of test cases. There are at most 20 test cases. This is followed by **n** test cases, each contains **m**+1 lines using the following format.

* The first line contains four integers: **P**, **S**, **C**, **m**. **P** is the number of people applying to the local contests, numbered from 1 to **P**. **S** is the number of sites. **C** is the number of contestants that each site can handle. (1 ≤ **P** ≤ 500; 1 ≤ **S** ≤ 20; 1 ≤ **C** ≤ 100)
* The next **m** following lines describe the sites that each contestant can participate. Each line contains 2 integers, the index of the contestant and the index of the local site that that contestant can join.

**Output**

For each test case, print the number of maximum contestants that the Programmers can accommodate to all of its local contest sites.

**Example**

|  |  |
| --- | --- |
| Input | Output |
| 22 2 1 41 11 22 12 24 3 1 121 11 21 32 12 22 33 13 23 34 14 24 3 | 23 |

|  |  |
| --- | --- |
| **J** | **Blanket** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

The winter is coming soon, and we have to keep people warm from harsh winter nights. There are **M** people in the town lying on a straight line at position 0, 1, 2, …, **M**-1. There are **n** sheets of blankets. Each sheet is infinitely long, but only some portions of it are thick enough to prevent you from freezing. Each blanket can be described by two integers (**a**, **b**). The blanket starts with a thick portion of length **a**, then a thin portion of length **b-a**, and this alternating pattern continues indefinitely. For example, if the blanket is given as (2, 3), then the first thick portion will cover persons at positions 0, 1, and the second thick portion will cover persons at positions 3, 4, and so on.

We lay all blankets altogether, all start at position 0. Depends on the pattern of each blanket, some lucky people will be covered by several thick portions from different blankets while unlucky one won’t be covered at all. Your task is to count the number of people that are covered by exactly 0, 1, 2, …, **n** thick blankets, respectively.

**Input**

The first line of input contains one integer **T**, the number of test cases (1 ≤ **T** ≤ 10). This is followed by T test cases, each uses the following format.

* The first line contains two integers **n** and **M** (1 ≤ **n** ≤ 105, 1 ≤ **M** ≤ 106) – the number of blankets and the number of people in town.
* Each of the next n lines contains two integers **a** and **b**, (1 ≤ **a** ≤ **b** ≤ 16), describing each blanket.

**Output**

For each test case, the output must contains **n**+1 integers, one per line, which are the number of people covered by 0, 1, 2, …, **n** thick blankets, respectively.

**Example**

|  |  |
| --- | --- |
| Input | Output |
| 13 302 53 53 6 | 6996 |

|  |  |
| --- | --- |
| **K** | **Concert Tour** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

Driven by recent fierce competition in retail market, The Store, one of the largest department store chains in Thailand starts a project to boost customer visiting. Since The Store has good relation with “Joy Boy”, a talented superstar, The Store decided that having “Joy Boy” perform concert once per month is the best way to lure more customer to the store. The project last **c** months, hence, there will be **c** Joy Boy concerts to be performed.

The Store has s department stores in different cities, one store per one city. Each city is numbered from 1 to **s**. The Store has to decide which store should have the concert on each month. The planning department of The Store has calculated, for every pair stores and months, the expected profit if Joy Boy performs a concert on a particular store in a particular month. Being nice and all, Joy Boy does not charge for his performance but he ask The Store to pay for his traveling and hotel expense. The planning department also knows the traveling cost for every pair of stores. Of course, it is possible to have Joy Boy stay at the same store for the next month and it might be some expense even though he performs at the same store.

Be noted that Joy Boy does not charge the cost traveling to the first concert. The Store asks you to project the maximum possible profits minus traveling cost from Joy Boy's Thailand tour for The Store.

**Input**

The first line in the input contain a single number, **n**, the number of test cases, 0 < **n** ≤ 10. Then, n test cases follow. For each test case, it contains (2\*s + 1) lines using following format.

* The first line contains 2 numbers, **s** and **c** where s is the number of stores and c is the number of concerts. 0 < **s** ≤ 100, 0 < **c** ≤ 50.
* The next s lines contain the projected profits. Each line for each store, starting from store 1 to store s. Each line contains ***c*** numbers – the projected profits for each month if Joy Boy would play at this store, starting from the first month to the c-th month. The profit is non-negative integer not exceeding 1,000.
* Finally, the last s lines contain the costs of moving from one store to other store. Each line for the cost of going from each store, starting from store 1 to store s**.** Each line contains s number – the cost of moving from the current store to each store, starting from store 1 to store s. The cost is non-negative integer not exceeding 100.

**Output**

For each test case, output one line with the maximum possible profit minus traveling cost.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 33 41 3 20 4050 20 1 220 50 50 10 10 1010 0 1010 10 03 320 20 2020 20 2020 20 200 20 4020 0 4040 10 02 410 20 10 2020 10 20 100 55 0 | 1706065 |

|  |  |
| --- | --- |
| **L** | **City** |
| **INPUT** | **STANDARD INPUT** |
| **OUTPUT** | **STANDARD OUTPUT** |

A group of urban planners are tasked with design of a new city. The new city is rectangle-shape city. The city will be divided into **N** rows and **M** columns of same-size square blocks. There are also several streets lying between these blocks. The table below represents the top-view map of one particular city having 4 rows by 6 columns of same-size square blocks. Each block is indicated by a star (‘\*’). The black borders between each block represent streets.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **\*** | **\*** | **\*** | **\*** | **\*** | **\*** |
| **\*** | **\*** | **\*** | **\*** | **\*** | **\*** |
| **\*** | **\*** | **\*** | **\*** | **\*** | **\*** |
| **\*** | **\*** | **\*** | **\*** | **\*** | **\*** |

In order to go from one block to adjacent block, one has to cross the street using a crosswalk. Each crosswalk must connect a pair of horizontally or vertically adjacent blocks. Moreover, there may be more than one crosswalk between each pair of blocks. We say that a crosswalk A belongs to a block B, if and only if, the crosswalk A connects the block B with any horizontally or vertically adjacent block of the block B.

We know the number of crosswalks that belongs to each block except only one block. Your task is to find the number of crosswalks of that block.

**Input**

 The first line contains a single integer **T** indicate the number of test cases (1 ≤ **T** ≤ 10). Then T test cases follow.

1. The first line of each test case contains two integers: **N** and **M**. The number of rows of blocks is **N** and the number of columns of blocks is **M** (3 ≤ **N**, **M** ≤ 500).
2. Next N lines contain the number of crosswalks that belongs to each block. Each line describes the crosswalks of a row of the grid, starting from the top row to the bottom row. In each line, there are M integers that describes the number of crosswalks belongs to each grid in that row, starting from the leftmost block to the rightmost block. The number of crosswalks for each block is a non-negative integer not exceeding 1,000. There will be exactly one block that is represented by -1. That block is the block that we don’t know the number of crosswalks.

**Output**

For each test case, print one line contains the number of crosswalks of the block that the number of crosswalks has not been counted.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 13 32 2 31 -1 31 1 0 | 1 |