The 2004 ACM ASIA Programming Contest Dhaka Site

Sponsored by IBM
Hosted by North South University
Dhaka, Bangladesh

7-8th October 2004
You get 15 Pages
8 Problems

Problem A

All Integer Average

Input File: a.in
Output: Standard Output

When we need to find the arithmetic average of some numbers we always tend to keep the result in a floating-point number as in many cases the average is a fraction. But if all the numbers are integers then we should not use floating-point numbers to store the average, as floating-point numbers can lead to all sorts of precision errors. In this problem your job is to find the average of some integers and express it in the form a *or* $\frac{b}{c}$ *or* $a\frac{b}{c}$, where a, b and c are all integers. In this problem we will denote this average as *avg***.**

a) If $|avg|>1$ and a fractional number then we should print it as $a\frac{b}{c}$.

b) If $|avg|<1$ and a fractional number then we should print it as $\frac{b}{c}$.

c) If *avg* is an integer we should print it as a.

d) If *avg* is negative then the whole result should be preceded by a minus (‘-’) sign and a space.

e) In the printed fractional part *b* and *c* should be relative prime. In other words *gcd(b,c)* should be 1.

f) The horizontal bar in the fraction should be formed using only the hyphen (‘-’). Its length should be equal to the number of digits in *c*. If the number of digits in *b* is less than *c* then *b* should be right justified on the horizontal bar.

Input

The input file contains less than **101** sets of input. Each set of input is given in a single line. The description of each line is given below:

The first integer **n(1≤n≤100)** of a line denotes how many numbers are to be averaged. It is followed by **n** numbers, all of which have absolute values less than **10000**.

Input is terminated by a case where **n=0**. This case should not be processed.

Output

For each line of input produce two or four lines of outputs. The first line contains the serial number of the output. The next one or three lines contain the value of the average following the rules specified above. The output lines should not contain any trailing spaces. And there must not be any leading or trailing spaces other than the ones that are required for formatting.

Sample Input

```
3 1 2 3
3 -1 -2 -3
3 1 2 4
4 2 4 6 10
3 -1 -2 -4
10 1 1 1 1 1 1 1 1 1 4
10 1 -1 1 -1 1 -1 -1 1 1 1
10 1 -1 1 -1 1 -1 -1 1 1 -3
0
```

Output for Sample Input

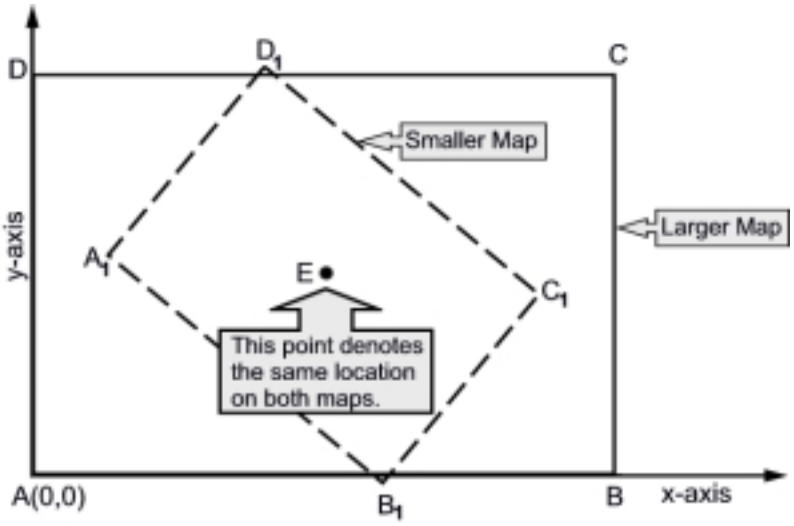
```
Case 1:
2
Case 2:
- 2
Case 3:
 1
2-
 3
Case 4:
 1
5-
 2
Case 5:
   1
- 2-
   3
Case 6:
  3
1--
 10
Case 7:
1
-
5
Case 8:
  1
- -
  5
```

Problem B

Overlaying Maps

Input File: b.in
Output: Standard Output

In the picture below you can see two rectangles **ABCD** and $\mathbf{A_1B_1C_1D_1}$. They are two rectangular maps of the same rectangular region. The size of the two maps is different but their aspect ratio is the same. This means $\mathbf{AB/A_1B_1=AD/A_1D_1}$.



Mathematicians say that there is always a geographical location on the bigger map such that the same location on the smaller map lies just straight above. So you can mark this location with one needle on both the maps at the same time. In the figure on the left, point **E** is such a point. Given the orientation of the two maps of the same rectangular area your job is to find this special point. You can assume that the bigger map is **ABCD** and the smaller map is $\mathbf{A_1B_1C_1D_1}$. You can also assume that point **A** is always at the origin, and one side of the bigger map is always in parallel with the x-axis. So obviously the other side is always in parallel with the y-axis.

Input

On the first line there will be **T (1≤T≤500)**, the number of test cases. Each case is represented by five lines. First of them will contain the coordinates of point **C**. The next four lines will contain coordinates of points $\mathbf{A_1}$, $\mathbf{B_1}$, $\mathbf{C_1}$ and $\mathbf{D_1}$ respectively. You can assume that **(10≤AB, AC≤100)** and $\mathbf{0.1\leq(A_1B_1/AB)\leq0.9}$.

Output

For each test case you must produce one line of output: the coordinates of the desired point with four digits after the decimal point. You can always assume that there will be one such point.

Sample Input	Output for Sample Input
2 8.0000000000 6.0000000000 2.0000000000 1.0000000000 6.0000000000 1.0000000000 6.0000000000 4.0000000000 2.0000000000 4.0000000000 34.0000000000000000 49.0000000000000000 18.2078241256558856 32.1681558770129171 22.7579122488961215 27.1147710637666363 30.0407315385745850 33.6722510060834468 25.4906434153343492 38.7256358193297276	4.0000 2.0000 26.6101 32.5721

Problem C

Reverse Assignment

Input File: c.in
Output: Standard Output

Alex has a simple assignment in his hand: counting the number of divisors of a given positive number **M**. For example the number **60** has **12** divisors **1**, **2**, **3**, **4**, **5**, **6**, **10**, **12**, **15**, **20**, **30** and **60**. Although it is very easy task, finding the reverse is not that easy. If you are given the number of divisors **D** of an unknown positive number **M**, it is not very easy to find **M** and in all cases there are more than one solution. The boring and easy assignments given by his teachers do not keep intelligent Alex interested for long. So he is trying to solve this rather difficult task now. Can you help him?

Input

The input file contains several lines of input. Each line contains an integer **D (0<D≤5000)**. Input is terminated by a line, which contains a number **0**. This line should not be processed.

Output

For each line of input except the last one you should produce one line of output. This line should contain the serial number of output followed by a positive number **M** less than $(10^{15}+1)$ whose number of divisors is exactly **D**. If there is no such number **M** less than $(10^{15}+1)$ whose total number of divisors is **D**, print the word **"Impossible"** without the quotes. If there is more than one possible value of **M** within the specified range, print the smallest one. Look at the output for sample input for details.

Sample Input	Output for Sample Input
3 4 12 60 4911 0	Case 1: 4 Case 2: 6 Case 3: 60 Case 4: 5040 Case 5: Impossible

Problem D

Zeros and Ones

Input File: d.in
Output: Standard Output

Binary numbers and their pattern of bits are always very interesting to computer programmers. In this problem you need to count the number of positive binary numbers that have the following properties:

- The numbers are exactly **N** bits wide and they have no leading zeros.
- The frequency of zeros and ones are equal.
- The numbers are multiples of **K**.

Input

The input file contains several test cases. The first line of the input gives you the number of test cases, **T (1≤T≤100)**. Then T test cases will follow, each in one line. The input for each test case consists of two integers, **N (1≤N≤64)** and **K (0≤K≤100)**.

Output

For each set of input print the test case number first. Then print the number of binary numbers that have the property that we mentioned.

Sample Input	Output for Sample Input
5 6 3 6 4 6 2 26 3 64 2	Case 1: 1 Case 2: 3 Case 3: 6 Case 4: 1662453 Case 5: 465428353255261088

Illustration

Here's a table showing the possible numbers for some of the sample test cases:

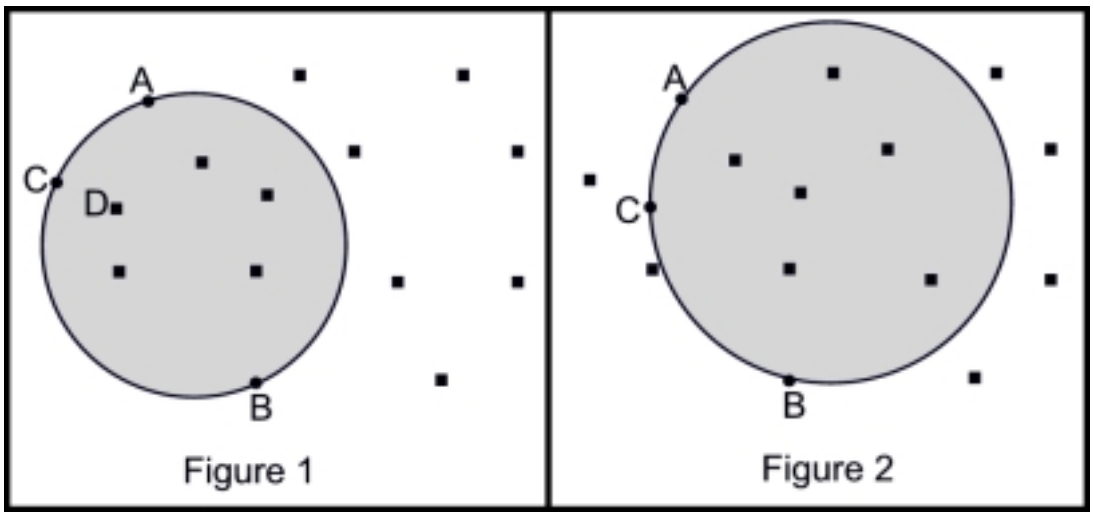
6 3	6 4	6 2
101010	111000 110100 101100	111000 110100 101100 110010 101010 100110

Problem E

Count the Points Inside

Input File: e.in
Output: Standard Output

We know that if three different points are not collinear then there is a unique circle that passes through them. When three points are collinear we get a unique straight line passing through them and a straight line can be considered as part of a circle of infinite radius. Given several thousand points and three special points **A**, **B** and **C** in a two dimensional Cartesian coordinate system your job is to find out how many points are contained in the circle passing through the three special points. Please note that your solution must be very efficient.



In the figure above the three special points are shown as small filled circles and normal points are shown as small filled squares. In Figure 1 we can see a circle passing through three special points **A**, **B** and **C**. There is also a normal point **D**. There are total five points strictly within the circle and three points are on the boundary. So the circle contains eight points in total. When the point **D** is turned into special point **C** we get another circle, which is shown, in Figure 2. This larger circle contains total nine points.

Input

The input file contains at most ten sets of input. The description of each set is given below.

First line of each set contains six integers $\mathbf{A_x}$, $\mathbf{A_y}$, $\mathbf{B_x}$, $\mathbf{B_y}$, **N** and **Q**. Here $\mathbf{(A_x, A_y)}$ is the coordinate of **A** and $\mathbf{(B_x, B_y)}$ is the coordinate of **B**, $\mathbf{N\ (0<N\le 50000)}$ is the total number of points (excluding **A** and **B**) and $\mathbf{Q\ (1\le Q\le 1000)}$ is the number of queries. Each of the next **N** lines contains two integers x_i, y_i, where $\mathbf{(x_i, y_i)}$ is the Cartesian coordinate of the **i**-th point $\mathbf{(1\le i\le N,\ -10000\le x_i,\ y_i \le 10000)}$. Each of the next **Q** lines contains one integer $\mathbf{S_C}$ $\mathbf{(1\le S_C\le N)}$, which indicates that the $\mathbf{S_C}$-th point is to be considered as point **C** for the current query. For example, in the first sample input the point with serial no **4** is **(5, 5)** as there are three points before it (**(-1, -1), (-1, -1)** and **(3,3)**) in the input

sequence. Please note that for a single set of input, point **A** and **B** are fixed. Only the position of point **C** is being altered for each query.

The input is terminated by a case where **N=Q=0**. This case should not be processed. You can see from the first sample input that more than one point can have the same coordinate. Such points should be considered as different points.

Output

For each set of input produce **Q+1** lines of output. The description of the output for each set is given below:

The first line contains the serial number of the output as shown in the output for sample input. Each of the next **Q** lines contains an integer **D** which indicates how many points will be within the circle passing through **A, B** and **C**. The points on the boundary of the circle are also considered inside the circle. So you should also consider **A**, **B** and **C** within the circle. If the radius of the circle is greater than **100000** then simply print the line **"Impossible"** instead of the integer **D**.

Sample Input	Output for Sample Input
0 10 10 0 6 2 -1 -1 -1 -1 3 3 5 5 -1 -1 12 16 4 1 0 10 10 0 8 2 -1 -1 2 2 4 4 6 6 8 8 12 12 14 14 16 16 3 5 0 10 10 0 0 0	Case 1: Impossible 7 Case 2: 8 7

:

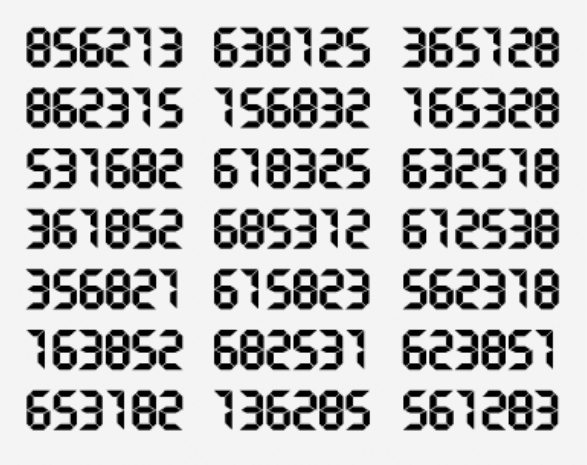
Problem F

Permutation Primes

Input File: f.in
Output: Standard Output

Permutations of a sequence of decimal digits have an interesting property. Any two permutations of a sequence of digits have a difference, which is divisible by **9**. For example:

|458967 – 456879| = 2088 = 9 * 232



We won't ask for the proof today (as it is very easy) but we will focus towards a different aspect of this property. There are some numbers whose difference with one (or more) of its permutations is of the form **9p**, where **p** is a prime less than **3333333**. These numbers are called permutation primes. For example **92-29=63=9*7**, where **7** is a prime. So, **92** is a permutation prime. Now you have to write a program that finds out the number of permutation primes within a specified range.

Input

First line of input contains an integer **T (0<T<201)** denoting the number of test cases to follow. Then follows **T** lines each of which contains two positive integers **p** and **q**. Both of them are less than **99999999**, without any leading zero(s) and **|p-q|<=1000**.

Output

There will be one line of output for each test case. At first print **"Case i: "** (without the quotes) where **i** is an integer denoting the **i-th** test case starting from one. Then the line will contain an integer **N** that denotes the number of permutation primes between **p** and **q** (inclusive).

Sample Input	Output for Sample Input
2 1 10 1 20	Case 1: 0 Case 2: 5

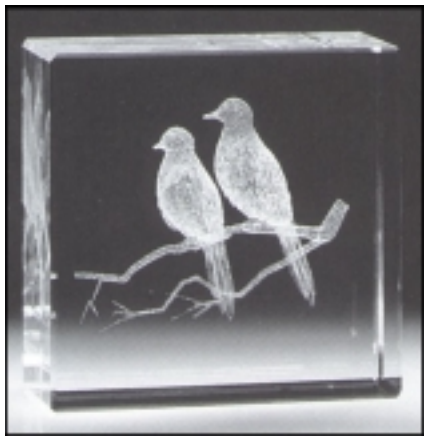
Note: **20-02=18=9*2**, so **20** is a permutation prime but **02** is not a permutation prime because a permutation prime does not have a leading zero.

Problem G

The Fairy Tale of ICPC

Input File: g.in
Output: Standard Output

It is now year **3004**, the **ACM ICPC** Programming Contest is still one of the largest and most prestigious Programming Contest of the World. This contest began almost **1030** years ago and many don't remember the history of this contest now. Some say that there is a wonderful story behind this contest. The story goes as follows:



Long long ago there lived a happy dove couple who loved each other very much. They were living peacefully but one day the female dove became very ill. The male dove began roaming around everywhere looking for proper medicine and found a wizard who knew the cure. But the wizard would only give him (male dove) the medicine if he could solve a problem for him, which was in the wizard's mind for a long time. As the problem was quite hard the male dove could not solve it. At that time a few very kind men came to his rescue. They gathered all the brilliant programmers from all over the world just to solve the wizard's problem, this gathering was known as the first **ACM ICPC** World Finals. The Wizard's problem was solved and the female dove got cured. Nice story! Isn't it? But the real problem is that now you have to solve this problem on your own.

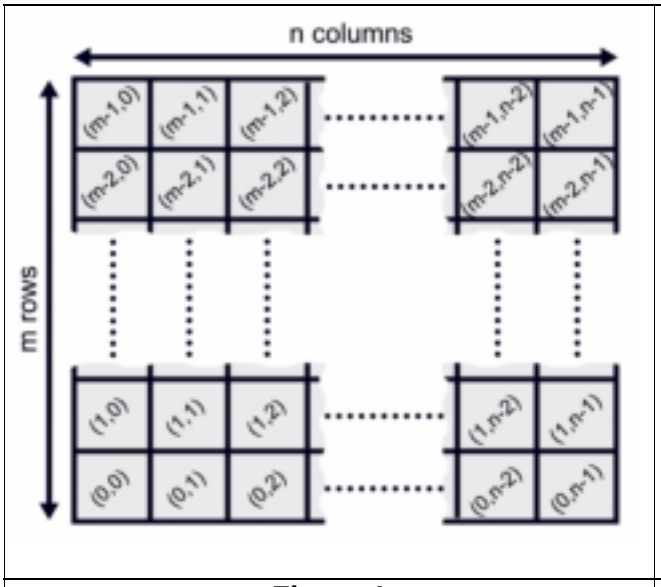


Figure 1

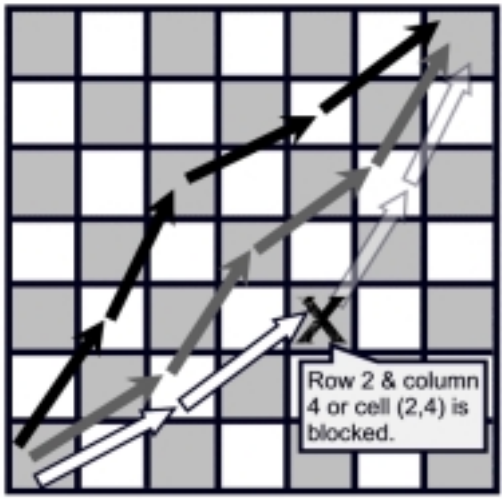


Figure 2

The wizard's problem was related to a chessboard. Given an **m*n** chessboard (**m**=number of rows and **n**=number of columns) one has to go from the lower left corner to the upper right corner. The

lower left corner is numbered as **(0,0)** and the upper right corner is marked as **(m-1, n-1)** as shown in **Figure 1**. But in each step one can only jump to another box that is **p** blocks away in the horizontal direction and **q** blocks away in the vertical direction or vice versa. No such moves, which would take one away from the destination, are allowed. So if the value **p=1** and **q=3** and one is in the **r**-th row and **c**-th column or location **(r, c)** then he can only jump to location **(r+1, c+3)** or **(r+3, c+1)** but he cannot jump to locations like **(r-1, c+1)** or **(r-1, c-1)** etc. In Figure **2** we can see a situation where **p=2** and **q=1** (or **p=1** and **q=2**). It shows two of the total six possible ways (considering that no square is blocked) of going from the lower left corner to the upper right corner and each of these two ways requires four steps to reach the destination. It also shows that the square at **(2, 4)** (row **2**, column **4**) is blocked, which has ruled out the possibility of a path through that square. Given the value of **m**, **n**, **p** and **q** and position of the squares which are blocked, your job is to find out the number of steps one requires to reach the destination and also the total number of possible ways to reach the destination.

Input

The input file contains less than **100** sets of inputs. The description of each set is given below:

Each line contains five integers **m, n (6<m, n<4001), p, q(0<p, q<10)** and **b(0≤b≤10)**. The meaning of **m**, **n**, **p** and **q** are described in the problem statement above. The integer **b** denotes the number of squares that are blocked. Each of the next **b** lines contains two integers **(r_i, c_i)** which denotes the row and column of the **i**-th blocked square.

Input is terminated by a set where the value of **m=n=0**. This set should not be processed. Please note that at least **90%** of the input test cases follow the limit **(6<m,n<300)**.

Output

For each set of input produce one line of output, which contains two integers **C** and **W**. Here **C** is the number of steps required to go from the lower left corner to the upper right corner and **W** is the total possible ways one can go from lower left corner to the upper right corner. If it is impossible to go from the lower left corner to the upper right corner with the given configuration then print a line **"Impossible"** instead as shown in the output for sample input. **W** can have at most **802** digits.

Sample Input	Output for Sample Input
8 7 1 2 2 2 1 1 2 8 8 1 3 0 49 49 2 1 0 93 109 3 1 1 24 16 108 50 3 1 6 85 47 45 23 54 34 24 40 40 24 82 46 0 0 0 0 0	Case 1: Impossible Case 2: Impossible Case 3: 32 601080390 Case 4: 50 64542614482000 Case 5: 39 376817

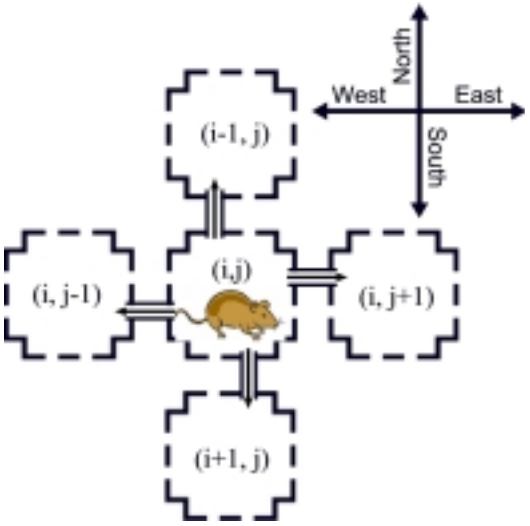
Problem H

Guinea Rats

Input File: h.in
Output: Standard Output

Rats like many other rodents are used in the labs for scientific experiments. But unlike many cruel surgical experiments our experiment on the rats would be a rewarding psychological one. We are trying to establish the fact that the rats have a good cognitive map in their brain – they have a good sense of directions. Here we would like to give an empirical proof to support our claim.

In our experiment, we build a maze for our rats. This maze is basically a rectangular grid of cells. There are tunnels that lead from one cell to another. Because of the rectangular arrangement, the tunnels can run in the four major directions only. From the cell at location **(i, j)** the rats may go East to **(i, j+1)**, West to **(i, j-1)**, North to **(i-1, j)** and South to **(i+1, j)** as shown in the figure on the right. These tunnels are directed. Thus a tunnel that takes a rat from cell from **(i, j)** to **(i, j-1)** not necessarily provides a path to get back to **cell (i, j)**. Some of these tunnels are open ended, if a rat goes through that tunnel, it would fall out of the maze. Now we allure the rats by putting mouth watering cheese cuts in some particular cells. Our experiment requires the rats to be trained to remember the sequence of **E** (East), **W** (West), **N** (North), **S** (South) moves that lead them to these cheese cuts.



We train a rat by releasing it in a designated start cell in the maze. Then it goes through the maze at its own will – taking the **EWNS** turns whimsically. But at some point in time it is bound to lose its vigor and stop moving. Then we take the rat out from the maze. If it stopped on a cell that contained a cheese cut we reward it with another cheese cut, leaving the original one in the maze; but if it stops on a cell that does not have any cheese cut, it goes unrewarded. One may wonder what would happen if the rat falls out of the board by following an open ended tunnel. Our explanation is simple, we put cheese cuts only in some particular cells in the maze – there is no cheese cuts in the world outside. So if a rat falls out it doesn't get any cheese.

We have seen that after training a rat several times in this way it seems to remember the rewarding move sequences correctly. Our experiment requires that we release the rat in the same starting cell and the cheese cuts are placed in exactly the same cells every time we put it in the maze. So if the rat finds that some **"EESSWNNS"** leads it to a cheese cut, it would always find a cheese cut with

that sequence of moves. Our smart rat only needs to differentiate the moves that are rewarding from the ones that are not. However, your task is not as simple as that.

We plan to train our rat in a maze and test its learning ability in a different maze. But to do that we need to make sure that these two mazes are identical. For our purpose identical mazes not necessarily mean identically constructed mazes. What we need is that they would be identical to the rats. **If a rat has a sequence of moves leading to a cheese cut in the training maze, it would also get a cheese cut with the same sequence of moves in the new maze. And if a sequence of moves leads it to nothing in the training maze, the same sequence must be unrewarding for the rat in the new maze.** This is where we need your help. Your task is to take the configuration of the two mazes as input, and tell us if they are identical from the rat's perspective.

Input

The input file contains several test cases. The first line of the input gives you the number of test cases, **T (1≤T≤25)**. Then **T** pairs of maze configuration will follow. The first line of a maze configuration starts with the dimension – the number of rows **R (1≤R≤20)** and the number of columns **C (1≤C≤20)** for the maze. Each of the next R lines would describe the C cells in that column. Each cell is represented by a 4-bit number (numbers in the range **0 to 15**). These numbers allows us to list all possible outgoing tunnels. The outgoing tunnels for North, East, South and West are represented by the 0^{th}, 1^{st}, 2^{nd} and 3^{rd} bits respectively. If a bit is set to zero then the tunnel in that direction is open ended – leading the rat out of the maze; whereas a bit set to one indicates that the rat can go to the next cell in that particular direction. We label the cells in the maze in row major order starting from **0**. After the description of the maze cells, the next line would give you the label of the starting cell for that maze. The first integer in the next line would give you the number of cells that would contain cheese cuts. Then that many cell labels will follow in the same line. You may find it helpful to relate the images in the illustration section to the second sample input.

Output

For each set of input print the test case number first. Then print **"Yes"** if our rat would find the pair of mazes to be identical, otherwise print **"No"**. The Sample Input/Output section will clarify the formatting issues.

Sample Input	Output for Sample Input
2 3 3 6 4 0 2 10 0 3 11 8 1 3 4 7 8 2 3 6 4 4 2 10 1 1 2 0 4 3 3 6 4 0 2 10 0 3 11 8 0 3 4 7 8 2 3 6 12 4 2 10 1 1 2 0 4	Case 1: Yes Case 2: No

Illustration

The following pictures illustrate the second sample test case. The numbers in the cells denote the cell labels, the rat shows the starting cell and the cheese cuts show the cells that would contain the cheese cuts. In the first maze the rat falls out of the maze if it takes a west bound move, thus fails to get any cheese. But in the second maze a west bound move would earn it a cheese cut.