A Airports

The government of a certain developing nation wants to improve transportation in one of its most inaccessible areas, in an attempt to attract investment. The region consists of several important locations that must have access to an airport.

Of course, one option is to build an airport in each of these places, but it may turn out to be cheaper to build fewer airports and have roads link them to all of the other locations. Since these are long distance roads connecting major locations in the country (e.g. cities, large villages, industrial areas), all roads are two-way. Also, there may be more than one direct road possible between two areas. This is because there may be several ways to link two areas (e.g. one road tunnels through a mountain while the other goes around it etc.) with possibly differing costs.

A location is considered to have access to an airport either if it contains an airport or if it is possible to travel by road to another location from there that has an airport.

You are given the cost of building an airport and a list of possible roads between pairs of locations and their corresponding costs. The government now needs your help to decide on the cheapest way of ensuring that every location has access to an airport. The aim is to make airport access as easy as possible, so if there are several ways of getting the minimal cost, choose the one that has the most airports.

Note: The input file is large; make sure your I/O code is fast.

Input

The first line of input contains the integer **T**(**T**<25), the number of test cases. The rest of the input consists of **T** cases.

Each case starts with two integers **N**, **M** and **A** (0<**N**<=10,000, 0<=**M**<=100,000, 0<**A**<=10,000) separated by white space. **N** is the number of locations, **M** is the number of possible roads that can be built, and **A** is the cost of building an airport.

The following **M** lines each contain three integers **X**, **Y** and **C** (1<=**X**, **Y**<=**N**, 0<**C**<=10,000), separated by white space. **X** and **Y** are two locations, and **C** is the cost of building a road between **X** and **Y**.

Output

Your program should output exactly **T** lines, one for each case. Each line should be of the form "Case #X: **Y Z**", where **X** is the case number **Y** is the minimum cost of making roads and airports so that all locations have access to at least one airport, and **Z** is the number of airports to be built. As mentioned earlier, if there are several answers with minimal cost, choose the one that maximizes the number of airports.

Sample Input	Sample Output
2 4 4 100 1 2 10 4 3 12 4 1 41 2 3 23 5 3 1000 1 2 20 4 5 40 3 2 30	Case #1: 145 1 Case #2: 2090 2

Problem setter: Muntasir Khan Special Thanks: Sohel Hafiz

B Big Number of Teams will Solve This

Writing up a code in Programming Contest for an easy problem isn't a big task but inexperienced contestants do not know that their output must match exactly with that of the judges. Very often they output extra spaces in their output because they feel it makes the presentation better. The requirement in reality is the output must match character per character with that of the judges. Submitted *runs* that give additional spaces are given the verdict "Output Format Error". If the output still mismatches after ignoring the spaces, it is given the verdict "Wrong Answer". A verdict of "Yes" is given when the output matches exactly. In this problem, you will have to determine the verdict for runs given the team output and the judges output.

Input

The first line of input consists of a positive integer **t<20,** where **t** denotes the number of test cases. Each case consists of two lines. The first line is the team's output and the second line is the judges output. Each line consists of at least one and at most **20** characters. The teams output consists of alphabets and spaces. The judges output consists of alphabets only.

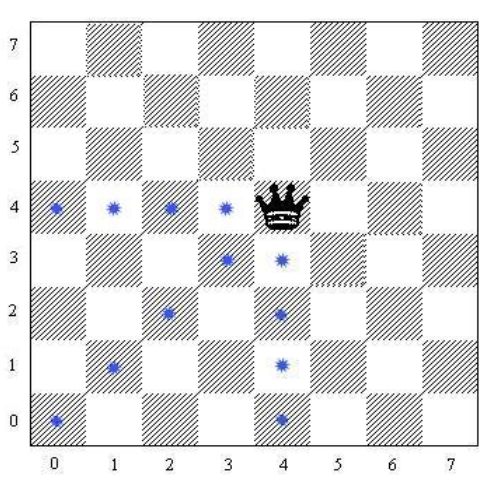
Output

For each case of input, there will be one line of output. It will first contain the case number followed by the verdict. Look at sample output for exact formatting.

Sample Input	Sample Output
3 yes yes Casematters casematters no space please nospaceplease	Case 1: Yes Case 2: Wrong Answer Case 3: Output Format Error

Problem setter: Shamim Hafiz Special Thanks: Md. Arifuzzaman Arif

C Corner the Queens



Corner the queen is a game played on n x n chess like board with two players. The rows and columns are numbered from 0 to n - 1. Then a queen is placed on a random cell other than (0, 0). Each player gives one move of the queen towards the cell (0, 0). The move is like a chess queen. As you know a queen can move any number of cells horizontally, vertically or diagonally. In Formal a player can move a queen from cell (a1, b1) to cell (a2, b2) if (a1 = a2 or b1 = b2 or |a1–a2| = |b1–b2|). Moreover in this game, move that takes queen away from the cell (0, 0) horizontally or vertically or diagonally is not allowed. Formally, if a player moves queen from cell (a1, b1) to (a2, b2) then (a2 ≤ a1 and b2 ≤ b1) must be held. The player who first reaches the cell (0, 0) is the winner. Now you may already have guessed if both the players play optimally, the starting position determines the winner. For some cell like (2, 0) player 1 always wins and for some cell like (1, 2) player 2 always wins.

In this problem we consider an infinite chess board for playing the game. A rectangular region is specified. A cell from that region will be picked randomly as a starting position for the queen. All you have to find is the probability that player 1 wins assuming that both players will play optimally.

Input

The first line of input will be a number **T** (**T** ≤ 15000) the number of test cases. Each of the following **T** lines will contain four integers **x1**, **y1**, **x2**, **y2** (0 ≤ **x1** ≤ **x2** ≤ 1000000, 0 ≤ **y1** ≤ **y2** ≤ 1000000). Here (**x1**, **y1**) is the lower left and (**x2**, **y2**) is the upper right portion of the rectangle. The lowest-leftmost cell is (0, 0) and it is always outside the given rectangle.

Output

For each line of input produce one line of output in the format "`Board X: n / d`". Here X is the number of case, n and d is the numerator and denominator of the probability expressed in reduced form. See the sample input and output for illustration.

Sample Input	Sample Output
3 1 0 2 2	Board 1: 2 / 3 Board 2: 1 / 1

1 0 7 0 1 2 1 2	Board 3: 0 / 1

Problem setter: Md. Towhidul Islam Special Thanks: Md. Arifuzzaman Arif

D Debugging RAM

You are working in a robot factory. You are responsible to write programs that run in the hardware of robots. You have implemented your first program into a robot and turn the robot on. But your robot is not acting according to your program. You have decided to debug your program. To do so, you have opened the robot's brain and created a copy of the whole RAM. The RAM contains different things like interrupt service routines, functions, variables etc. You are interested to look into what the variables hold at a particular time. But since it is a tedious task, you have decided to write another program to simplify your task.

Input and Output

There will be several test cases in the input file. The first line of each test case starts with two positive integers **b** <= **8** and **v** <= **200**, where **b** is the number of bits in a byte and **v** is the total number of variables in the RAM. These two integers will be separated by a single space. Each of the next **v** lines will contain a variable name, $\mathbf{s_i}$, and number of bytes that variable will occupy in RAM, $\mathbf{t_i}$, **1** <= **i** <= **v** and **1** <= $\mathbf{t_i}$ <= **8**. A variable name consists of lower case and upper case letters and the length of a variable name is not more than **20**. Variable names are case sensitive. It is guaranteed that no two variable names are same in a set and all variables are unsigned. From the next line contents of the RAM will start. There will be exactly $\sum_{i=1}^{v} t_i$ lines and each line will have exactly **b** characters. Each character in the RAM is either 0 or 1. First t_1 lines are value for the variable s_1, next t_2 lines are value for the variable s_2 and so on. RAM contents are aligned as most significant byte first. Next line will have single integer, **1** <= **q** <= **v**, number of variables for which you need to output the content of variable. Next **q** lines will each have a variable name. These variable names may or may not be one of the $\mathbf{s_i}$'s. If it is one of the $\mathbf{s_i}$, output the variable name, and equal sign "=" (quotes for clarity only), and contents of the variable (in unsigned decimal number). If the queried variable name is not one of those $\mathbf{s_i}$'s, then output the variable name and equal sign only.
For clarification, see sample input and output.

Sample Input	Sample Output
4 4 ab 2 Cd 2	Ab= Cd=245 ef=2730

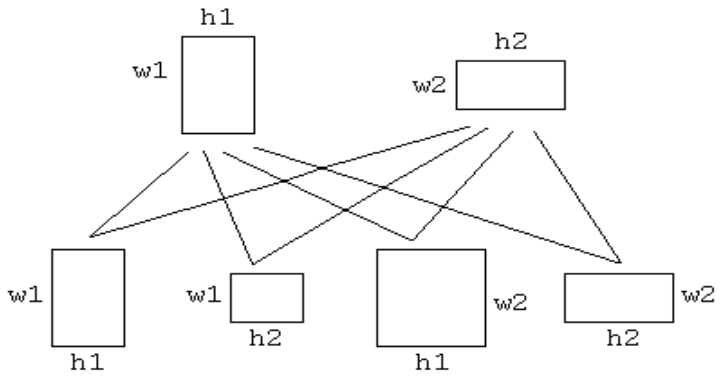
ef 3 gh 1 0101 1111 1111 0101 1010 1010 1010 1000 4 Ab Cd ef gh	gh=8

Problem setter: Adnan Chowdhury Special Thanks: Shamim Hafiz

E Extreme Primitive Society

Do not be misled by the name. Primitive society is a community of very advanced intelligent unisexual beings, called the primitives. The reason that they are called primitives is that they are all primitive geometric shapes, rectangles. Every member of the primitive society have two genetic traits, **width(w)** and **height(h)**, both can be expressed as positive integers. The primitive society have in fact so advanced in science and technology that, now they reproduce only through the means of genetic engineering. Their reproduction system works as follow.

Two primitives deposit their gene samples in a Generation Creating Machine (GCM). The GCM then performs genetic engineering tasks to produce up to **4** offspring. Say the parent primitives had genetic traits **(w1,h1)** and **(w2,h2)** respectively. Then the **4** offspring would have genetic traits **(w1,h1)** , **(w2,h2)** , **(w1,h2)** and **(w2,h1)** respectively.
After this, a mutation is performed but this is optional and GCM decided this randomly. The mutation process is defined as the incrementing or decrementing of **h** value by **1** and / or incrementing or decrementing of **w** value by **1.** And thus, a new generation of primitives is born. See fig. for better understanding of the procedure.



Due to their scientific advancements, the primitives have become immune to everything except the weather change. Therefore, they do not need to reproduce very often to ensure survival of their species. Once in ten years or so they deposit their gene samples to the GCM. Say there are **n** people in the current generation, then the GCM uses the genes from every individual of the current generation with every other individual performing **n(n-1)/2** mating. The GCM can also apply mutation on some of the resulting offspring to increase their fitness. A primitive is considered absolutely fit if both of its genetic traits have same value, i.e. the rectangle is a square.

Note that, although **n(n-1)/2** mating are performed, at most **2n(n-1)** offspring are born in the next generation. Why up to **2n(n-1)**, not exactly **2n(n-1)** ? Because they don't have any need for people with redundant genetic configurations and the GCM makes sure that only one person of each possible genetic configuration will remain in the society.

Given an initial population of primitives, your job is to calculate the minimum number of generations should pass before an absolutely fit individual is born in this society.

Input

There will be multiple test Cases, not more than **1010**. Each case starts with an integer **n(>=2)** on a line by itself. This is the number of individuals in the initial population. Following will be n lines, each having **2** positive integers representing the genetic configuration of each individual of the society. All numbers will be positive integers not exceeding **100**.

Output

For each test case of input, print one line of output beginning with **"Case x : "** where x is the test case number. This text should be followed by the minimum number of generations required before an absolutely fit individual is born in this society.

Sample Input	Sample Output
3 35 40 30 35 32 44 3 35 68 70 1 79 25	Case 1 : 1 Case 2 : 1

Problem setter: Raiyan Kamal Special Thanks: Sabbir Yousuf

F Few Teams will Solve This

You have just finished a compiler design homework question where you had to find the parse tree of an expression. Unfortunately you left your assignment in the library, but luckily your friend picked it up for you. Instead of e-mailing you the parse tree so that you can rewrite the solution, your friend decides to play a practical joke and sends you just the DFS and BFS trace. Rather than trying to redo the entire question you decide to reconstruct the tree. But doing that you realize there could be more than one tree that produces the given DFS and BFS trace.

Can you find out how many trees produce the given DFS and BFS trace?

Input

The input file contains several test cases as described below.

The first line of the input is the number **n** (1 <= **n** <= 1000) of nodes in the tree. The nodes in the tree are numbered 1, 2, ..., **n**. The remaining numbers are the BFS traversal followed by the DFS traversal. Note that when a parent was expanded the children were traversed in ascending order. The last line of input is a case where **n** = 0 and that doesn't need to be processed.

Output

For each case, output the case number first followed by the required result. Since the result could be very big, output the result modulo 19821202.

Sample Input	Sample Output
8 4 3 5 1 2 8 7 6 4 3 1 7 2 6 5 8 3 1 2 3 1 2 3 0	Case 1: 1 Case 2: 2

Problem setter: Rujia Liu Special Thanks: Renat Mullakhanov/Sohel Hafiz

G Giving Candies

Two siblings are trying to figure out a way of sharing a packet of candy. They would like to divide it as equally as possible. The problem is that they are all attached into one continuous strip. Sharing it would involve breaking the strip into smaller pieces containing one or more individual candies. The candies themselves are not all the same, thus in a totally fair distribution, each sibling must get an equal number of the same type of candy.

But, being small children, they also have some additional (and more annoying) demands. Each sibling must get exactly one whole segment broken off from the main strip, and that one segment must contain all the candies that form his share. The child will not accept more than one segment for his or her share. What is more, the order of the candies from left to right in one child's segment must be exactly the same as that in other child's (and rotating the strip is not allowed), otherwise they have difficulty verifying that they have got a fair distribution. So both segments must be exactly the same. To make matters worse, there are certain types of candies that neither sibling wants. Those candies cannot be in the segment handed to the children, because they consider any segment containing those candies to be ruined.

Your job is, given a large strip of candy and a list of 'bad' candies, identify the largest contiguous segment that occurs at least twice (the two occurrences cannot overlap, obviously) and does not have any bad candies in it. Any left over candy is discarded. If there are no such segments in the strip, the whole strip is discarded and the children get nothing.

Input

The first line of input contains a single integer **N** (**N**<=50). **N** cases follow. Each case consists of exactly two lines. The first line is a non-empty string (1<=length<=1000) describing one large strip of candy. Each candy is given a unique character that is either a lower case or upper case English letter ('a' to 'z' or 'A' to 'Z', both ranges inclusive). The second line is a string of characters corresponding to 'bad' candies. Like the first line, this string consists of upper or lower case English letters. But in some cases there may not be any bad candies (i.e. there are no candy types that the children don't like), and the second line will be an empty line instead. Furthermore, each candy can occur only once in the list of bad candies (no duplicates).
Each candy corresponds to exactly one unique letter, so upper case and lower case forms of the same character denote different candies (i.e. 'a' and 'A' are not the same candy).

Output

Your program should print exactly **N** lines of output, one for each input case.

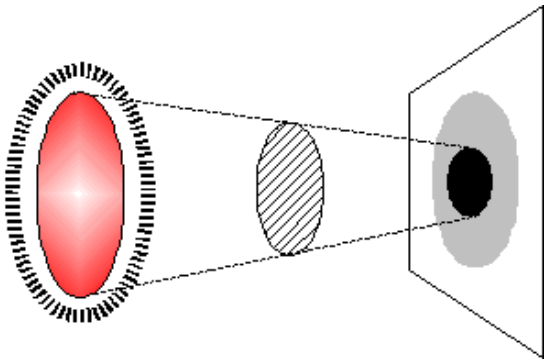
Each line should be of the form "Case #X: Y", where X is the case number and Y is the size of the largest segment that at least two non-overlapping occurrences and has no bad candies in it. If there is no such segment, Y is zero.

Sample Input	Sample Output
6 ABADZEDGBADEZ ZEG CaNdY baaaa a aaaab b MnMnMn Aa aAaA fghFGH	Case #1: 3 Case #2: 0 Case #3: 0 Case #4: 2 Case #5: 2 Case #6: 2

Problem setter: Muntasir Khan Special Thanks: Md. Arifuzzaman Arif

H How Dark can it be?

Little Dipto is fascinated by shadows. He spends hours holding up various objects in front of the wall, watching the shadow grow and shrink as he moves the object. But these days he has been developing a special interest in shadows of polygons; convex polygons to be precise. But he cannot seem to be able to predict the size of the shadow before he actually cuts out a paper polygon and forms a shadow of it on the wall. He has now turned to you for help.



Input

The input consists of at most 60 test cases. Each test case starts with an integer **N**, the number of vertices in the polygon given. The value of **N** satisfies 3 <= **N**<= 10000. The next **N** lines contain a pair of integers each, x and y, specifying the coordinates of each vertex of the polygon in counterclockwise order (the z-coordinate of each vertex is always 0). These coordinate values will always be between -10000 and 10000 inclusive. The last line consists of two real numbers 0 <= **R** <= 100 and 1 <= **D** <= 1000. These two numbers specify the light source: it is a flat circular source of uniform intensity along the plane z = D, centred on (0,0,D) and radius R. The wall is a flat, smooth and infinitely large surface along the plane of z = -D. You may assume that there is no other light source in the vicinity. You may also assume that the input data will always be such that it specifies a polygon of non-zero area and that the same point will not be present in the data multiple times.

The last input is followed by a single 0 on the line, which should not be processed.

Output

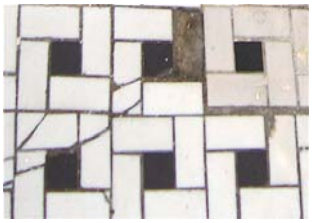
There should be one single line of output for each test case. It should be formatted as "Case c: u p", where c is the case number with the first case designated as case 1, u is the area of the umbra (the part of the shadow that is completely dark, usually near the center) and p is the area of the rest of the shadow (the lighter part of the shadow that is only partially dark) called penumbra. The diagram above should help clarify the terms. Both areas should be within a relative error bound of 1e-6 of the actual value. See the sample output below for further clarification.

Sample Input	Sample Output
5 0 0 10 0	Case 1: 100.000000 770.681952

10 10 1 10 0 9 5 10 0	

Problem setter: Samee Zahur Special Thanks: Md. Towhidul Islam

I Ignore the Blocks



Today is Tilliby's 9th birthday. He has been receiving all kinds of gifts from people like glow-in-the-dark stickers, electronic calculator, a new toothbrush and so forth. However, this funny looking puzzle involving domino tiles caught his attention. The rules were as follows – the puzzle consisted of a rectangular grid of R-by-C cells, and it must be completely filled by 2x1 sized tiles, of which there is sufficient supply. No two tiles may overlap. To complicate things even more, certain cells of the grid is marked unusable, which must not be covered by any of the tiles. All other cells, however, must be covered by exactly one tile. Now in spite of all these complications, this was an easy exercise for Tilliby. However, when it came to counting the number of ways this can be solved, even his new and shiny calculator could not help him. Can you?

Input

The input consists of at most 100 test cases. Each test case starts with a line containing 3 integers, **R**, **C** and **N**. This line will be followed by **N** other lines, each containing two integers, $\mathbf{r_i}$ and $\mathbf{c_i}$, where $\mathbf{r_i}$ is the row position of the $\mathbf{i^{th}}$ unusable cell and $\mathbf{c_i}$ is the column position. All these input integers will be within the following ranges: 1 <= **R** <= 4, 1 <= **C** <= 100000000, 0 <= **N** <= 100, 0 <= $\mathbf{r_i}$ < **R**, 0 <= $\mathbf{c_i}$ < **C**.

The last test case will be followed by a single line containing three 0 (zeroes) which should not be processed.

Output

The output for each test case should consist of one single line of the form "Case c: x", where c is the serial number of the test case starting from 1, and x is the number of ways the specified tiling can be performed. Since this number can be very large, output its value in mod 10000007.

Sample Input	Sample Output
2 2 0 2 4 0 2 4 2 0 0 0 3 0 0 0	Case 1: 2 Case 2: 5 Case 3: 1

Problem setter: Samee Zahur Special Thanks: Md. Arifuzzaman Arif