Kung Fu Panda Revisited

A

Input file: stdin
Output file: stdout
Peoblem Code: panda

Master Shifu: So, you are the dragon warrior, hmm?

Po: Umm... I guess so!

Master Shifu: Wrong! You are not the dragon warrior. You would never be the dragon warrior... until you have learnt the secret of the dragon scroll!

Po looking at the scroll: Wow! So how does this work? Do you have a ladder or... any trampoline?

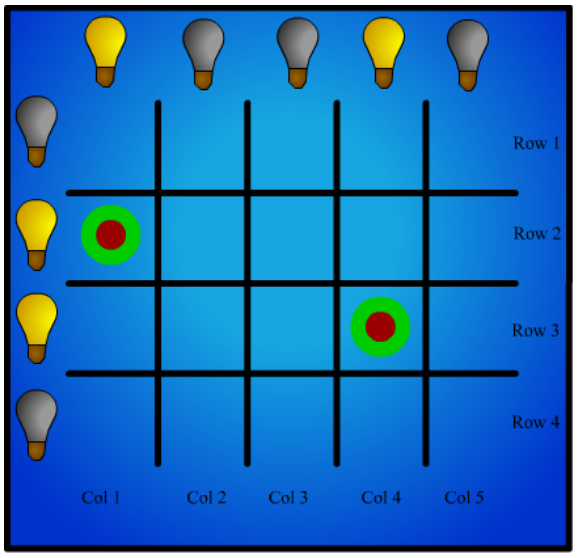
Master Shifu chuckling: Do you think is that so easy that I just hand you the secret to the limitless power?

Po: Umm... Ya.. I mean No.. I mean...

Master Shifu: You have to win the ancient sacred game of Su against Viper, Monkey, Mantis, Crane and Tigress, The Furious Five!!!

Po gave a big smile and then what! He fell down- senseless.

How long he was sleeping he did not know. When he woke up he heard a big cheering around. He discovered him on a side of a giant square board. Master Shifu was describing the rule of the game- This board is $\mathbf{50 \times 50}$ in size, and it is just like a giant uncolored chess board. Each column and each row has a bulb. When one touches a square the corresponding row and column bulb will toggle. That means, if it was ON before touching the square then it will be OFF and vice versa. Initially Master Oogway will lit some lights. You have to touch squares as less as possible and turn all the lights OFF. You can never touch the same cell twice. For example, in the picture you can see the row **2** and row **3** bulbs are ON and column **1**, column **4** bulbs are ON. So Po can finish the game by pressing **2** switches, at $\mathbf{(2, 1)}$ and $\mathbf{(3, 4)}$. Note that, the board is $\mathbf{500 \times 500}$ but for space constrained we showed here just a small part of the board.



As the game began The Furious Five finished the game just in few seconds. Po was thinking how to win the game. He has to solve the problem in minimum move to win the tournament. He was thinking and thinking.. Then suddenly he felt his power, his biggest power, the power of hunger. He soon imagined a big Sushi as his prize of this game and guess what, he won the game!

Input

First line of the input denotes the number of test case (not more than **100**).

For each test case there are two lines of non negative integers. First integer of the first line denotes number of row light bulbs that are ON. Then the indices of the row of the ON light bulbs follow. Similarly the description of the ON column light bulbs follows in the second line. No row index or column index will appear more than once. Each index will be between **1** to **500** inclusive. The board size will always be $\mathbf{500 \times 500}$ as stated in the problem description. You may assume that in all cases number of ON lights

in rows will be same as number of ON lights in column.

Output

For each case, print a line with test case. If the problem is impossible to solve then print ***Impossible*** in next line. Otherwise print a positive integer (note that in this problem **0** is not considered to be a positive integer), minimum number of touch Po requires. Then you have to print the cell indices. If there are multiple possible minimum solution output the lexicographically smallest one. Index of a cell is denoted by "r, c" where r is the row index of the square and c is the column index. One **solution A** is lexicographically smaller than another **solution B** if first s squares in both the solutions are same, and $s+1$th square is not same and say $s+1$th square of **solution A** is "rA, cA" and $(r+1)$th square of **solution B** is "rB, cB" and then $(rA < rB$ or $(rA = rB$ and $cA < cB))$.

See the sample input/output for exact formatting.

Sample input and output

stdin	stdout
4 1 20 1 11 1 1 1 20 2 1 2 2 1 2 2 2 3 2 1 4	Case 1: 1 20, 11 Case 2: 1 1, 20 Case 3: 2 1, 1 2, 2 Case 4: 2 2, 1 3, 4

Explanation of case 3: There is another solution touching 2 squares. "1, 2" "2, 1". But the solution above is lexicographically smaller than this one.

The last test case is the one explained in the problem statement.

The Story of Spring Secret

B

Input file:	stdin
Output file:	stdout
Peoblem Code:	secret

In the Programmers' Mythology, the story of Spring Secret is very widely known. Though the actual identity of it is not very well defined in the myth which plays a major role to make it such a huge secret. Some says it is related to the seasons while some other thinks it might be referring to a natural fountain. Of course, there are a few who believe that it might well be a person as well. But as it has been an unsolved mystery and a wonder for such a long time, let's stay away from the definition which may end up only confusing us.

Though it is unsure what this Spring Secret actually is, it is well known that this mysterious Spring Secret has a great involvement with the hilsha fish. It is said that you will find the term 'hilsha' a good several times in any texts related to Spring Secret. Let us not concern ourselves with this Hilsha fondness of Spring Secret for now. Rather we shall analyze some texts from the original Spring Secret myth and try to find how many 'hilsha' we can find.

Input

The input file beings with an integer $\boldsymbol{case}$ $\boldsymbol{(1 \leq case \leq 60)}$ denoting the number of test cases. $\boldsymbol{case}$ test cases follows. Each case has two lines. The first line is an integer $\boldsymbol{n}$ $\boldsymbol{(1 \leq n \leq 100)}$, denoting the number of words in the text of this case. The next line contains $\boldsymbol{n}$ space separated words. The words consist of lower case English characters only.

Output

For each test case, print a line in the format, "Case X: Y", where $\boldsymbol{X}$ is the case number and $\boldsymbol{Y}$ is the number of times the word 'hilsha' is found in the text.

See the sample input/output for exact formatting.

Sample input and output

stdin	stdout
2 5 abc hilsha def hilsha hilsho 2 hilsha hilshaa	Case 1: 2 Case 2: 1

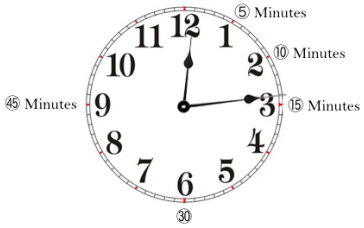
Find the Correct Time

C

Input file: stdin
Output file: stdout
Peoblem Code: clock

John has a beautiful clock. He likes his clock very much. But it has a simple problem. The hour hand has same length as the minute hand. As John is very intelligent, he can usually figure out the time. For example at **12**'o clock he will understand the time because both hands are at same place. On the other hand at **4**'o clock he will understand the time because the hour hand at 12 and minute hand at **4** is invalid time. But consider $\mathbf{01:10\frac{70}{143}}$, John will not be able to identify the time because the clock will look exactly same at $\mathbf{02:05\frac{125}{143}}$ too. Now John wonders how many different times in a day he will not be able to figure out the time. Note that John only cares about the time range when he is awake.

So John wants your help. He wants you to write a program that will find number of time when he will not be able to figure out the time. He wants you to write a generalized program that does not assume there is **60** minutes in full rotation of minute hand and and **12** hours in full rotation of hour hand. You know, some alien may face similar problem but has different hours/minutes in a full rotation of hour/minute hand.



Input

Input file starts with a line containing an integer $\boldsymbol{T} \leq \mathbf{1000}$. $\boldsymbol{T}$ lines follow. Each line describes a test case.

Each test case consists of two integers $\boldsymbol{a}$ and $\boldsymbol{b}$ followed by two time $\boldsymbol{s}$ and $\boldsymbol{e}$. You may assume that $\mathbf{1} < \boldsymbol{a}, \boldsymbol{b} < \mathbf{100}$. Assume that both $\boldsymbol{s}$ and $\boldsymbol{e}$ occurs in same day and the time $\boldsymbol{s}$ occurs strictly before the time $\boldsymbol{e}$ and time difference between $\boldsymbol{s}$ and $\boldsymbol{e}$ is less than $\boldsymbol{a}$ hours. A full rotation of minute hand is $\boldsymbol{b}$ minutes and a full rotation of hour hand is $\boldsymbol{a}$ hours. John is awake in the range $\boldsymbol{s}$ to $\boldsymbol{e}$ inclusive. The times have $\boldsymbol{HH:MM}$ format. Where $\boldsymbol{HH}$ represents the hour and will be less than $\boldsymbol{a}$ and $\boldsymbol{MM}$ represents the minute and will be less than $\boldsymbol{b}$. Note that for this problem we are ignoring the concept of $\boldsymbol{am/pm}$.

Output

For each test cases output a line formatted as "Case $\boldsymbol{i}$: $\boldsymbol{x}$" . Where $\boldsymbol{i}$ is the case number $\boldsymbol{x}$ is the number of time when he will not be able to figure out the time. Note that to figure out the time John will only use the position of clock hands. That is he does not use the fact that he is awake in this moment.

See the sample input/output for exact formatting.

Sample input and output

stdin	stdout
5 12 60 00:00 11:59 12 60 05:00 05:59 11 99 02:31 09:78 12 60 01:10 01:11 12 60 01:09 01:10	Case 1: 132 Case 2: 11 Case 3: 75 Case 4: 1 Case 5: 0

Monkeys on Twin Tower

D

Input file: stdin
Output file: stdout
Peoblem Code: tower

Being inspired by the ongoing popularity of animation films, the monkeys are trying to be smarter. They have realized that the only way to get smarter is to learn mathematics. Hence, they have started to do so. With the creative brains as they have, they are applying math in all aspects of life. Now, one of these mathematician monkeys are standing in front of a multi-storied twin tower. The twin tower is actually a couple of tall buildings standing parallel to each other. Each of the buildings have n floors. The ground floor is ***floor* 0**, the next one is ***floor* 1** and so on. So, there are $2n$ floors in total in the twin tower. Each of these floors have a fruit inside it. The monkey knows in advance the amount of time required to eat the fruit in any floor. The monkey starts from the ground floor, climbs up toward the top of the buildings and has to eat exactly n fruits. From floor i, he has only two ways to go to floor $i+1$. He can go the floor $i+1$ of the same building that he is on in floor i. As he is a good jumper, it can be completed in no time. Also, he can go to floor $i+1$ of the other building using a spiral stair connecting the two buildings. This will take a certain time. Please note that, the monkey can only move to floor $i+1$ from floor i. He wants to figure out the minimum time required to eat n fruits. Can you verify how good his math is?

Input

The input file begins with an integer ***case*** $(1 \leq case \leq 60)$, the number of test cases in the input file. This line is followed by ***case*** test cases. Each test case consists of **5** lines. The first line has a single integer n $(1 \leq n \leq 1000)$, the number of floors in each building. The second line contains n integers separated by a single space. These integers denote the number of seconds required to eat the fruit in each floor for the left building. The time is given in ascending order of the floor i.e. the first integer is the number of seconds required to eat the fruit in ground floor of the first building while the last integer is the time required for the fruit in the topmost floor. The next line, containing n integers, describe the same values for the right building. Each of the above $2n$ integers has a value between **1** and **100**. The line **4** has $n-1$ space separated integers. These values denote the time required to jump from the left building to the right one. So, the first integer is the number of seconds to jump from ground floor left building to ***1st floor*** right building. Finally, the fifth line contains $n-1$ more integers giving the time required for jumping from the right building to the left one. The jumping times have values between **1** and **50**.

Output

For each test case, print a line in the format, "Case X: Y", where X is the case number and Y is the minimum number of seconds required to eat n fruits.

See the sample input/output for exact formatting.

Sample input and output

stdin	stdout
1 4 5 6 8 9 7 9 3 10 5 2 3 2 4 3	Case 1: 26

Wishing Snake

E

Input file: stdin
Output file: stdout
Peoblem Code: snake

'Wishing Snake' is a computer game for children. In this game, there is a big board and a snake. The board contains **1000** check points numbered from **0** to **999**. And the snake can go from any checkpoint to another without crossing other checkpoints. Initially the snake is at checkpoint **0**.

Now n children come in front of the board and they start to wish. Each wish is like "I want to see the snake walking from checkpoint u to checkpoint v." And the snake can fulfill this wish by walking from checkpoint u to checkpoint v without crossing any other checkpoints. Each child can have multiple wishes. At first a child comes in front of the board and makes his wishes. After that a new child comes and makes his new wishes (that means not wished by any children yet). And it continues until the nth children.

After that the snake starts walking from one checkpoint to another. It can only walk from one checkpoint to another if any of the children had wished it. The snake wants to fulfill all the wishes done by all the children. Since you are the main designer of the game; you want to find whether it's possible or not.

Input

Input starts with an integer $T(\leq 65)$ denoting the number of cases.

Each case starts with an integer $n(1 \leq n \leq 100)$. Then for each child an integer $k(k \geq 0)$ is given. Each of the next k lines contains two different integers u v $(0 \leq u, v < 1000$ and $u \neq v)$ denoting that the child wants to see the snake going from checkpoint u to checkpoint v. You may assume that all the wishes are distinct and correct and the total number of wishes in any case is between **1** and **10000** (inclusive).

Output

For each case, print the case number and '**YES**' if it's possible, otherwise print '**NO**'.

See the sample input/output for exact formatting.

Sample input and output

stdin	stdout
2 2 2 0 9 9 10 1 10 15 1 2 0 9 0 11	Case 1: YES Case 2: NO

Warning: Large input file. You may need to use faster input methods, e.g. scanf.

Grid Coloring

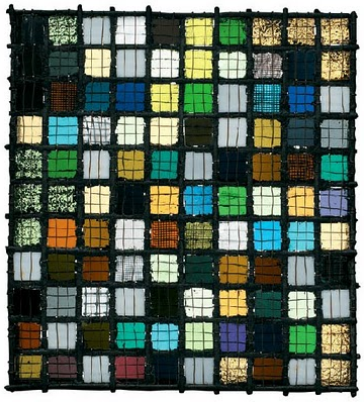
F

Input file: stdin
Output file: stdout
Peoblem Code: grid

You have to color an $N \times M$ two dimensional grid. You will be provided K different colors for this. You will also be provided a list of B blocked cells of this grid. You cannot color those blocked cells. A cell can be described as (x, y), which points to the yth cell from the left of the xth row from the top.

While coloring the grid, you have to follow these rules

1. You have to color each cell which is not blocked.
2. You cannot color a blocked cell.
3. You can choose exactly one color from K given colors to color a cell.
4. No two vertically adjacent cells can have the same color, i.e. cell (x, y) and cell $(x + 1, y)$ cannot contain the same color.



You have to calculate the number of ways you can color this grid obeying all the rules provided.

Input

Input starts with an integer T, the number of test cases. T is around 600. Each test case starts with a line containing four integers N $(1 \leq N \leq 1,000,000)$, M $(1 \leq M \leq 1,000,000)$, K $(0 \leq K \leq 1,000,000)$ and B $(0 \leq B \leq 500)$. Each of the next B lines will contain two integers x and $y(1 \leq x \leq N, 1 \leq y \leq M)$, the row and column number of a blocked cell. Each of these B lines is distinct.

Output

For each test case, print a single line in output in the format "Case I: C" (quote for clarity). Here "I" is the case number and "C" is an integer the number of ways for coloring the grid modulo $1,000,000,000$.

See the sample input/output for exact formatting.

Sample input and output

stdin	stdout
3 3 3 3 0 3 4 4 2 3 1 3 3 2 2 5 2 1 2 2 2	Case 1: 1728 Case 2: 186624 Case 3: 20

Warning: Large input file. You may need to use faster input methods, e.g. scanf.

Another Toy Story

G

Input file: stdin
Output file: stdout
Peoblem Code: lottery

Little John loves toy very much. So much that he decided to buy toys with all the money he earned in this summer. As a matter of fact he earned k Rupees during his summer vacation. Now he has found a shop where every toy has same price, x Rupees. So he decided to use his money to buy as many toys as possible. There are n different toys in the shop. To his utter disappointment John has found he does not have enough money to buy all the toys. Now he is wondering how to choose which toy to buy.

Then suddenly he has an idea. He has a really intelligent toy (which he bought earlier), which takes a number as input and puts another number as output. If he gives it an input m it will show him a random number between **1** and m (inclusive). Unfortunately it takes y seconds for this toy to compute a random number. He has total t seconds available to buy toy (The shop will close then). He came up with following two algorithms.

Algorithm money

1. Number the toys uniquely using the numbers **1** to n arbitrarily.
2. Repeat step 3 to 5 until the money available is less than x.
3. Use the toy to choose a random number z between **1** and n.
4. If toy number z is already bought then go to step 3
5. Buy toy z and go to step 3.

If he uses **Algorithm money** he may need more time than he has.

Algorithm time

1. Number the toys uniquely using the numbers **1** to n arbitrarily.
2. Repeat step 3 to 5 until the time remaining is less than y.
3. Use the toy to choose a random number z between **1** and n.
4. If toy number z is already bought then go to step 3
5. Buy toy z and go to step 3.

If he uses **Algorithm time** he may need more money than he has.

Now because of potential problems with both algorithms (he may run out of time or money), he wants to compare them to check which one is better. He asks you to help him in this task (he may let you play with his toy). John wants you to write a program that will compute, given amount of money available k and amount of time available t what is the expected time needed if he uses **Algorithm money** and what is the expected cost if he uses **Algorithm time**.

Input

Input file starts with a line containing an integer $T \leq 10000$. T lines follow. Each line is description of a test case.

For each case the input consists of **5** integers, n, k, t, x and y separated by one or more spaces. You may assume $0 < n, k, t \leq 10^5$, $0 < x, y \leq 100$ and $n \times x > k$.

Output

For each test case output a line formatted as "Case i: a b" (without quotes) where i is the case number, a is the expected time needed if he uses **Algorithm money** and b is the expected cost if he uses **Algorithm time**. Print a and b rounded to **2** digits after decimal point. There will be no case in input where changing the output by 10^{-6} will change the rounded answer. Follow the sample output.

Sample input and output

stdin	stdout
7 2 1 2 1 1 5 10 10 3 4 5 4 8 1 1 10 100 4 11 1 100000 99999 99999 1 1 100000 99999 1 1 99 100000 1 99999 99 1	Case 1: 1.00 1.50 Case 2: 15.67 5.40 Case 3: 6.42 4.16 Case 4: 19.29 37.83 Case 5: 1109014.61 63211.87 Case 6: 109792446.69 0.00 Case 7: 0.00 6257975.32

Counting Perfect BST

H

Input file: stdin
Output file: stdout
Peoblem Code: bst

BST is the acronym for Binary Search Tree. A BST is a tree data structure with the following properties.

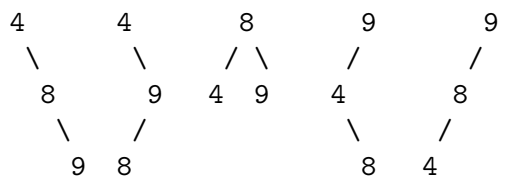
i) Each BST contains a root node and the root may have zero, one or two children. Each of the children themselves form the root of another BST. The two children are classically referred to as left child and right child.

ii) The left subtree, whose root is the left children of a root, contains all elements with key values less than or equal to that of the root.

iii) The right subtree, whose root is the right children of a root, contains all elements with key values greater than that of the root.

An integer $\boldsymbol{m}$ is said to be a perfect power if there exists integer $\boldsymbol{x} > \mathbf{1}$ and $\boldsymbol{y} > \mathbf{1}$ such that $\boldsymbol{m} = \boldsymbol{x^y}$. First few perfect powers are $\{\mathbf{4, 8, 9, 16, 25, 27, 32, 36, 49, 64, 81, 100, 121, 125, 128, 144}\}$. Now given two integer $\boldsymbol{a}$ and $\boldsymbol{b}$ we want to construct BST using all perfect powers between $\boldsymbol{a}$ and $\boldsymbol{b}$ inclusive, where each perfect power will form the key value of a node.

Now, we can construct several BSTs out of the perfect powers. For example, given $\boldsymbol{a} = \mathbf{1}$ and $\boldsymbol{b} = \mathbf{10}$, perfect powers between $\boldsymbol{a}$ and $\boldsymbol{b}$ are $\mathbf{4, 8, 9}$. Using these we can form the following five BSTs.



In this problem, given $\boldsymbol{a}$ and $\boldsymbol{b}$, you will have to determine the total number of BSTs that can be formed using perfect powers between $\boldsymbol{a}$ and $\boldsymbol{b}$ inclusive.

Input

First line of input file contains an integer $\boldsymbol{T}$. $\boldsymbol{T} \leq \mathbf{20,000}$ test cases follow. Each case of input contains two integer $\boldsymbol{a}$ and $\boldsymbol{b}$ ($\mathbf{1} \leq \boldsymbol{a} \leq \boldsymbol{b} \leq \mathbf{10^{10}}$ and $\boldsymbol{b} - \boldsymbol{a} \leq \mathbf{10^6}$) as defined in the problem statement.

Output

For each case of input, there will be one line of output. It will first contain the case number followed by the total number of distinct BSTs that can be formed by the perfect powers between $\boldsymbol{a}$ and $\boldsymbol{b}$ inclusive. The values may be arbitrarily large, therefore all answers must be given modulo **100000007**.

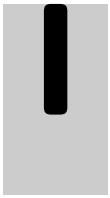
See the sample input/output for exact formatting.

Sample input and output

stdin	stdout
3 1 4 5 10 1 10 1 3	Case 1: 1 Case 2: 2 Case 3: 5 Case 4: 0

Warning: Large input file. You may need to use faster input methods, e.g. scanf.

Finding Genes in DNA



Input file: `stdin`
Output file: `stdout`
Peoblem Code: `dna`

Your friend is a biologist. He has just sequenced a DNA and wants to know about contribution of different genes in that DNA. Both Gene and DNA can be represented by a sequence of letters or strings. Given the sequence of a DNA $\boldsymbol{D}$ and a Gene $\boldsymbol{G}$, your friend uses following method to calculate the contribution.

- Generate a list $\boldsymbol{P}$ of proper[1] non-empty prefixes of $\boldsymbol{G}$ and another list $\boldsymbol{S}$ of proper non-empty suffixes of $\boldsymbol{G}$. Additionally let $\boldsymbol{L}$ is the list of all strings that is concatenation of a prefix and a suffix. So if $\boldsymbol{G} = \boldsymbol{ACCT}$ then $\boldsymbol{P} = \boldsymbol{A, AC, ACC}$, $\boldsymbol{S} = \boldsymbol{T, CT, CCT}$ and $\boldsymbol{L} = \boldsymbol{AT, ACT, ACCT, ACT, ACCT, ACCCT, ACCT, ACCCT, ACCCCT}$. If $|\boldsymbol{G}| = \boldsymbol{n}$ then it is obvious that size of $\boldsymbol{L}$ is $\boldsymbol{(n-1)^2}$.
- For each element of $\boldsymbol{L}$, count the number of time it occurs as substring in $\boldsymbol{D}$. Contribution of Gene $\boldsymbol{G}$ in DNA $\boldsymbol{D}$ is total of these values. For example if $\boldsymbol{D} = \boldsymbol{ACTACCTACCCCT}$ then

AT	0
ACT	1
ACCT	1
ACT	1
ACCT	1
ACCCT	0
ACCT	1
ACCCT	0
ACCCCT	1
Total	6

As this process is very clumsy he wants to automate this process. As he is not a programmer, he needs your help. He will be very grateful if you kindly write him a program which given sequence of the DNA and the Gene will calculate contribution of the Gene in the DNA.

Input

Input file starts with a line containing an integer $\boldsymbol{T} \leq \mathbf{40}$. $\boldsymbol{T}$ test cases follow. Each test case description consists of **2** lines.

First line of each test case is the sequence of DNA and second line is the sequence of the Gene. The length of these strings are less than **50000** and consists of only $\boldsymbol{A}$, $\boldsymbol{C}$, $\boldsymbol{T}$ and $\boldsymbol{G}$.

Output

For each test case output a line formatted as "Case i: a", where $\boldsymbol{i}$ is the case number and $\boldsymbol{a}$ is the contribution as described in problem statement.

See the sample input/output for exact formatting.

[1]Proper prefix (suffix) of a string $\boldsymbol{S}$ is a prefix (suffix) of length smaller than $|\boldsymbol{S}|$. Here $|\boldsymbol{S}|$ denotes length of $\boldsymbol{S}$.

Sample input and output

stdin	stdout
3 ACTACCTACCCCT ACCT AAA AAAA AAAA AAA	Case 1: 6 Case 2: 4 Case 3: 8

Warning: Large input file. You may need to use faster input methods, e.g. scanf.

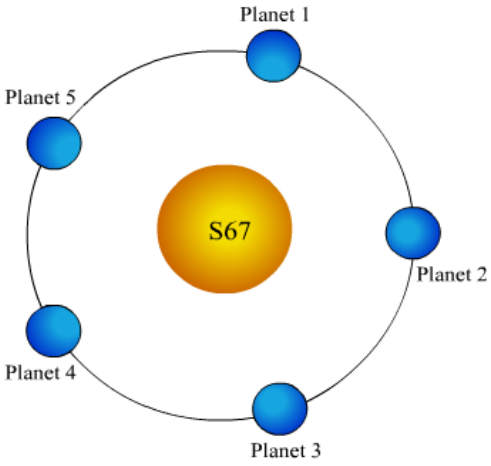
Chimpanzee Management

J

Input file: stdin
Output file: stdout
Peoblem Code: chimp

In 35th century earth became uninhabitable for human civilization because of pollution. They live in several planets orbiting around a star named S67. Those planets are quite different from the planets in our solar system. In our solar system, planets have their own orbit. But there these planets move following the same orbit. When one wants to go from one planet to another they can just go to adjacent planet because if they go near to the star S67 their spaceship gets melted, so they do not take risk and just moves from one planet to adjacent planets only. As the planets are in a closed orbit, each planet has two adjacent planets. Every day at 9AM two spaceships leave every planet going to two adjacent planets and they reach their destination at 18AM. (In those planets 50hours make a day). These spaceships wait for 1 hour and start their returning voyage at 19AM and reach at their own planets at 3PM. You may assume that planets are numbered from $\mathbf{1}$ to $\boldsymbol{n}$ consecutively, for each $\mathbf{1} \leq \boldsymbol{i} < \boldsymbol{n}$ Planet $\boldsymbol{i}$ and Planet $\boldsymbol{i}+\mathbf{1}$ are adjacent and Planet $\mathbf{1}$ and Planet $\boldsymbol{n}$ are adjacent. The picture shows the orientation for $\mathbf{5}$ planets. Note that, here a passenger from Planet $\mathbf{4}$ may travel to Planet $\mathbf{1}$ in one day. Say two spaceships come from Planet $\mathbf{1}$ and Planet $\mathbf{4}$ to Planet $\mathbf{5}$ at 18AM. Then the passenger from Planet $\mathbf{4}$ rides on the returning spaceship for Planet $\mathbf{1}$ at 19AM. A spaceship will not leave its own planet if there is no passenger, however it does not cancel its returning voyage even if there is no passenger.

One day Academy of Chimpanzee Management (ACM) decided that, they will equally distribute all the Chimpanzees of all the planets. ACM decides which Chimpanzee will go where. However they know that they can not send more than one Chimpanzee at one time. It is because, a chimpanzee inhales too much oxygen, and the oxygen cylinder attached with a spaceship is not enough for more than one Chimpanzee. ACM wants to accomplish this task as soon as possible.



Your task is: Given the number of planets and the number of Chimpanzees in each planet. You are to find minimum number of days required to equally distribute the Chimpanzees. Sometimes because of some technical reasons, there may be several pair of planets $\boldsymbol{a}$ and $\boldsymbol{b}$ such that there can not be voyage between them.

Input

First line contains number of test cases, $\boldsymbol{T}(\leq \mathbf{50})$. Hence follow $\boldsymbol{T}$ test cases.

In the first line of every test case there are two non negative integers, $\boldsymbol{N}$ and $\boldsymbol{M}(\leq \mathbf{3})$ denoting number of planets and number of forbidden pair respectively. $\left(\mathbf{0} < \boldsymbol{N} \leq \mathbf{10}^{(\boldsymbol{M}+\mathbf{2})}\right)$.

Then there follows $\boldsymbol{N}$ integers $\boldsymbol{i}$th of which denotes number of Chimpanzees in $\boldsymbol{i}$th planet. Total number of Chimpanzees will not exceed $\mathbf{10}^{\mathbf{9}}$.

Then there follows $\boldsymbol{M}$ pair of integers "$\boldsymbol{a}$ $\boldsymbol{b}$" denoting the $\mathbf{1}$ based index of the planets between which voyage is forbidden. ($\mathbf{1} \leq \boldsymbol{a}, \boldsymbol{b} \leq \boldsymbol{N}$, and a and b are adjacent two planets).

Output

For every test case, output case number in "Case x:" format where $\boldsymbol{x}$ is the test case number, then there follows the minimum number of days to accomplish the task after a space. If it is impossible, then output "Impossible".

See the sample input/output for exact formatting.

Sample input and output

stdin	stdout
3 3 0 1 2 3 3 1 1 2 0 2 3 2 0 1 0	Case 1: 1 Case 2: 1 Case 3: Impossible

Warning: Large input file. You may need to use faster input methods, e.g. scanf.

An Evil Plan

K

Input file: `stdin`
Output file: `stdout`
Peoblem Code: `plan`

Three little prince Akor, Bonsur and Conmor are playing in the garden. In this game they stand in three positions to form a triangle and they pass a magic ball to each other.

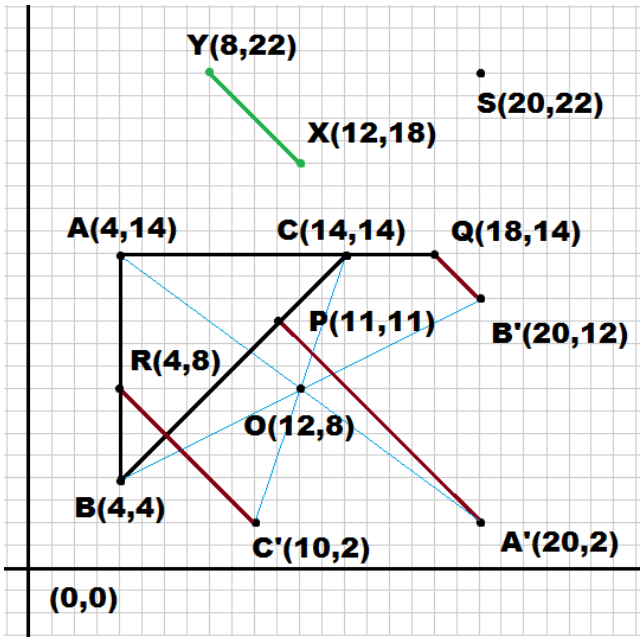
Their father, King Dimosor, is aware of the fact that the garden may not be safe for them, since the enemies can take this as an advantage. But he wants to see the kids smiling; so he permitted them to play.

Meanwhile, the enemy King Elohan wants to kill the princes. So, he asked his astrologer, who suggested, "There is a lucky Orchid flower in the garden; an archer can kill a prince if he can find a position where he can fire his arrow towards the prince through the Orchid, keeping the Orchid exactly in the middle between his bow and the prince." So, King Elohan sent three archers to do the job.

Now, King Dimosor has captured the astrologer and found the evil plan. So, he calculated the expected positions of the archers and called three guards - Pinocio, Qurota and Rubic. He planned to give three bombard canons to the guards such that they can kill the archers. But there is only one problem. The canons can be fired according to the wind direction; either in the wind direction or in opposite to the wind direction. Otherwise it would be tough for the guards to find the correct angle to shoot.

Since the guards know the wind direction, Pinocio took the position such that he forms a line with Bonsur and Conmor; and he can also shoot the archer who might fire at Akor. Qurota took the position which forms a line with Akor and Conmor; and he can shoot the archer who might target Bonsur. And Rubic took the position which forms a line with Akor and Bonsur and he can also shoot the archer who might fire at Conmor.

After a while, the guards saw the archers and fired the canons, and the archers were killed. But the princes were quite shocked hearing the sound of the heavy firing of the canons. That's why they want to run away from the guards as far as they can. There is a Sunflower in the garden and the princes love the flower so much that they want to run in a circular path centering the Sunflower and they will run together so that from their initial positions they all cover the same angular distance with respect to the Sunflower.



In this problem, all the positions are in a $\mathbf{2}\boldsymbol{D}$ plane. And for simplicity each one including the flowers is just a point in the plane. To find their farness with the guards, each prince first finds the Euclidean distance between his position and the $\mathbf{3}$ lines formed by (Pinocio, Qurota), (Pinocio, Rubic) and (Qurota, Rubic). Then from all their calculated distances, they take the minimum distance, which they call the farness. They want to maximize this farness. Since they don't know how to find such complex thing, they seek your help.

In the picture, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{O}$ and $\boldsymbol{S}$ denote the positions of Akor, Bonsur, Conmor, the Orchid and the Sunflower respectively. $\boldsymbol{A'}$, $\boldsymbol{B'}$ and $\boldsymbol{C'}$ denote the archer positions who aim at Akor, Bonsur and Conmor respectively. $\boldsymbol{P}$, $\boldsymbol{Q}$ and $\boldsymbol{R}$ denote the positions of Pinocio, Qurota and Rubic respectively. $\boldsymbol{X}$ to $\boldsymbol{Y}$ denotes the wind direction.

$\boldsymbol{PA'}$, $\boldsymbol{QB'}$ and $\boldsymbol{RC'}$ are parallel to $\boldsymbol{XY}$ (wind direction) and these lines indicate the guards aiming at the archers. Now you are given the coordinates of $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{P}$, $\boldsymbol{R}$ and $\boldsymbol{S}$. You have to find the maximum farness as described above.

Input

The first line of input will contain an integer $\boldsymbol{T} \leq \mathbf{1000}$ denoting the number of cases. Each case contains **7** lines, each line containing two real numbers denoting a point ($\boldsymbol{x}$ co-ordinate followed by $\boldsymbol{y}$ co-ordinate). The lines will contain the coordinates of $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{P}$, $\boldsymbol{R}$ and $\boldsymbol{S}$ respectively. You may assume that $\boldsymbol{ABC}$ forms a valid triangle and $\boldsymbol{XY}$ is not parallel to any of $\boldsymbol{AB}$, $\boldsymbol{AC}$ or $\boldsymbol{BC}$. You may safely assume both co-ordinates of $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{O}$, $\boldsymbol{P}$, $\boldsymbol{Q}$ and $\boldsymbol{R}$ are between $\mathbf{-1000}$ to $\mathbf{1000}$ inclusive.

Output

For each case print the case number and the maximum farness rounded to **3** decimal places.

See the sample input/output for exact formatting.

Sample input and output

stdin	stdout
1 4 4 14 14 12 18 8 22 11 11 4 8 20 22	Case 1: 16.565

The test case corresponds to the above picture.