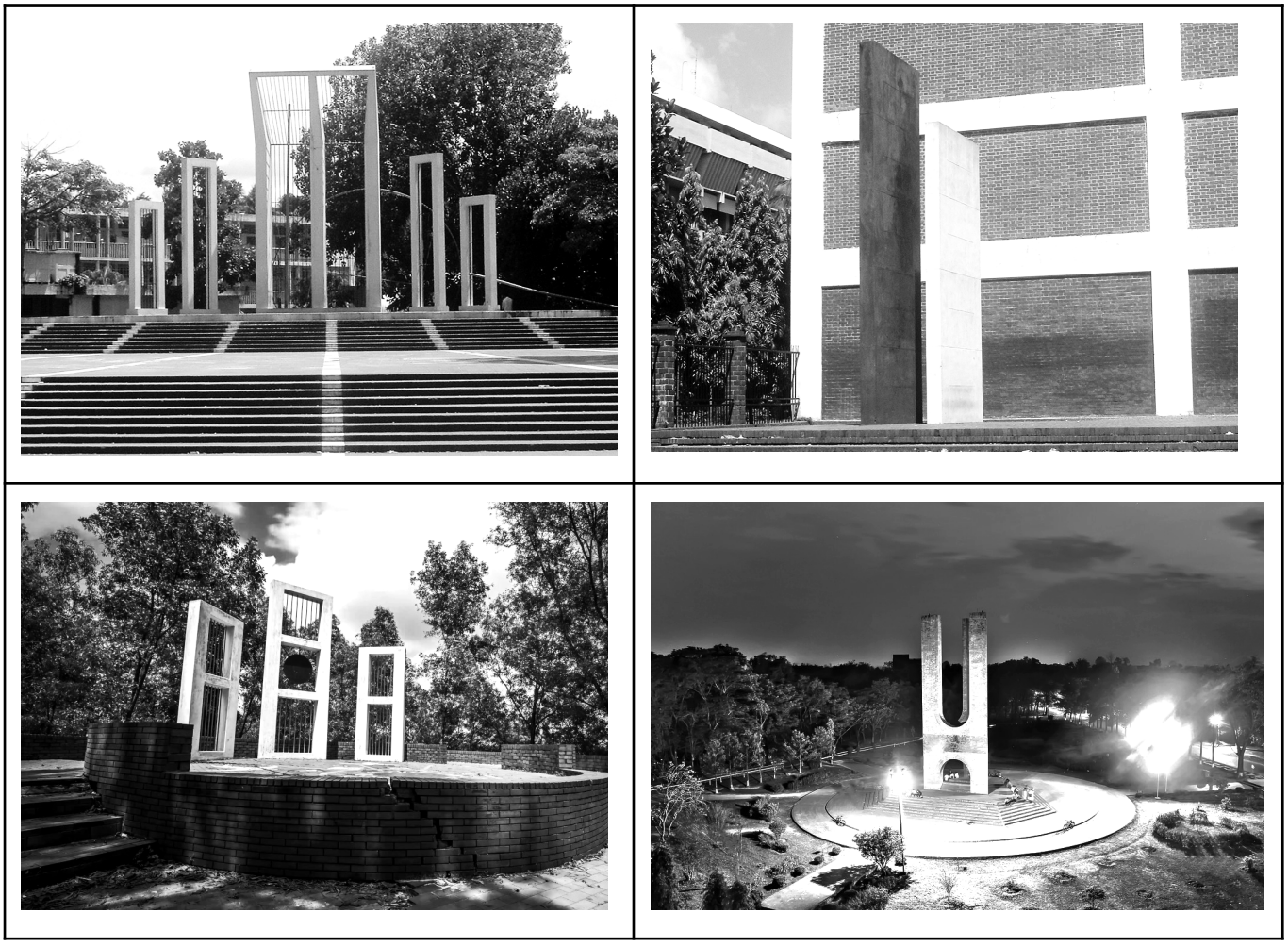
icpc.foundation
icpc International Collegiate Programming Contest
Daffodil International University
icpc global programming tools sponsor

Official Problemset
ICPC Dhaka Regional 2018

```
do
{
    printf("Do not touch anything\n");
} while("Not asked to touch");
```



You will get:
10 Problems, 16 Pages, 300 Minutes

Problem A

Input: Standard Input
Output: Standard Output

Average of Combination

Given **N** numbers (1, 2, 3 … N), you can take **R** different numbers in several ways. For example, if **N = 4** and **R = 3**, there are 4 possible ways:

- 1, 2, 3
- 1, 2, 4
- 1, 3, 4
- 2, 3, 4

If we fix **N**, and vary **R**, we will get many different combinations. In total, for a particular **N**, there can be $2^N - 1$ different combinations. Let's define two statistics **S** and **A** for a combination, where **S** is the sum and **A** is the average of the numbers in the combination. For your convenience, these values (R, S and A) are shown below for all the combinations of **N = 4**:

R	Combinations	Sum (S)	Average (A)
1	1	1	1
	2	2	2
	3	3	3
	4	4	4
2	1, 2	3	1.5
	1, 3	4	2
	1, 4	5	2.5
	2, 3	5	2.5
	2, 4	6	3
	3, 4	7	3.5
3	1, 2, 3	6	2
	1, 2, 4	7	2.33…
	1, 3, 4	8	2.66…
	2, 3, 4	9	3
4	1, 2, 3, 4	10	2.5

You can see that, the average of different combinations may vary. We are interested in ordering the combinations according to their average (non-increasing order). If two combinations have the same average, then the one with fewer numbers will come earlier in order. If two combinations have the same average and the same number of elements then the lexicographically (considering the combinations as sorted sequence of integers) smaller one will come earlier in order. So for **N = 4**, the combinations in order are:

1. 4
2. 3, 4
3. 3
4. 2, 4
5. 2, 3, 4
6. 1, 3, 4
7. 1, 4
8. 2, 3
9. 1, 2, 3, 4
10. 1, 2, 4
11. 2
12. 1, 3
13. 1, 2, 3
14. 1, 2
15. 1

For this problem, given **N** and **K**, you have to print the **K^{th}** (1-based) combination of the numbers in the range from **1** to **N** in the order mentioned above.

Input

First line will contain number of test cases, **T (1 ≤ T ≤ 100)**. Each case will contain two integers **N (1 ≤ N ≤ 100000)** and **K (1 ≤ K ≤ min (2 × 10^7, 2^N-1))**.

Output

Print case number and then the required combination in each line, separated by spaces. Please see the sample for clarification.

Sample Input	Output for Sample Input
4 4 7 4 15 4 1 10 30	Case 1: 1 4 Case 2: 1 Case 3: 4 Case 4: 4 7 8 9 10

Problem B

Input: Standard Input
Output: Standard Output

Counting Inversion

The number system we are used to is called the decimal number system. The digits of decimal number system are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

$$(123)_{10} = 1 \times 100 + 2 \times 10 + 3 \times 1 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$

The leftmost digit is called the most significant digit and the rightmost one is called the least significant digit. Let, **K** be a decimal integer represented as $\mathbf{d_n \ldots d_2 d_1 d_0}$, where, $\mathbf{d_n}$ ($\mathbf{d_n > 0}$) is the most significant digit (the leftmost digit) and $\mathbf{d_0}$ is the least significant digit (the rightmost digit).

Any two digits of **K**, $\mathbf{d_i}$ and $\mathbf{d_j}$ form an inversion **if and only if,** $\mathbf{d_i > d_j}$ where $\mathbf{i < j}$.

We define the number of digit inversions of **K** as **DI(K)**. For example, if **K = 123,** then, $\mathbf{d_2 = 1, d_1 = 2, d_0 = 3}$ and **DI(K) = DI(123) = 3** (as $\mathbf{d_0 > d_1}$, $\mathbf{d_1 > d_2}$ and $\mathbf{d_0 > d_2}$). If **K = 253,** then $\mathbf{d_2 = 2, d_1 = 5, d_0 = 3,}$ then **DI(K) = DI(253) = 2** (as $\mathbf{d_0 > d_2}$ and $\mathbf{d_1 > d_2}$). Similarly, **DI(5) = 0 , DI(321) = 0, DI(491383) = 6**.
In this problem, you are given two integers **x** and **y** in the decimal number system.
You have to calculate $\sum_{K=x}^{y} DI(K)$.

Input

Input starts with an integer **T($1 \le T \le 50000$)** denoting the number of test cases.
Following **T** lines each contains two integers **x** and **y** (without leading zeros) where **($1 \le x \le y \le 10^{14}$)**. The dataset is huge, so use faster I/O methods.

Output

For each test case, the output should contain the case number in the format: "**Case T:** ", where **T** is the test case number followed by the desired answer in a single line. Please see the sample for clarification.

Sample Input

```
5
1 9
1 100
50 60
23 2343
345 99373
```

Output for Sample Input

```
Case 1: 0
Case 2: 36
Case 3: 4
Case 4: 6083
Case 5: 410008
```

Problem C

Input: Standard Input
Output: Standard Output

Divisors of the Divisors of an Integer

The function **d(n)** denotes the number of positive divisors of an integer **n**. For example **d(24) = 8,** because there are **8** divisors of **24** and they are **1, 2, 3, 4, 6, 8, 12** and **24**. The function **sndd(n)** is a new function defined for this problem. This denotes "The summation of number of divisors of the divisors" of an integer n. For example,

sndd(24) = d(1) + d(2) + d(3) + d(4) + d(6) + d(8) + d(12) + d(24) = 1 + 2 + 2 + 3 + 4 + 4 + 6 + 8 = 30.

Given the value of **n,** you will have to find **sndd(n!)**, here **n!** means factorial of **n**. So **n! = 1 × 2 × 3 × ... × n**.

Input

The input contains at most **1000** lines of test cases.
Each line contains a single integer that denotes a value of **n ($1 \le n \le 10^6$)**. Input is terminated by a line containing a single zero.

Output

For each line of input produce one line of output. This line contains an integer that denotes the modulo **10000007 (or $10^7 + 7$)** value of **sndd(n)**.

Sample Input	Output for Sample Input
4 5 0	30 90

Problem D

Input: Standard Input
Output: Standard Output

Faketorial Hashing

Are you familiar with polynomial hashing? If you are not, all the better! You don't need to know what polynomial hashing is, the world is better off without it. I hate polynomial hashing so much that I found a new way to hash strings. It is called the ***Faketorial Hashing***.

First, let's define a function, *ord(ch)* = the position of *ch* in the alphabet + 1, where **ch** can be any lowercase letter. So, *ord(a) = 2, ord(b) = 3, ord(c) = 4, … ord(z) = 27*.

Let *fact(x)* be **x!** or the factorial of **x**. A few examples, *fact(1) = 1, fact(2) = 2, fact(3) = 6, fact(4) = 24, fact(5) = 120*, etc.

Given a string **S** of length **N,** consisting of lowercase letters only, the ***Faketorial Hashing*** of **S**, is defined as below:

fake_hash(S) = fact(ord(S[0])) × fact(ord(S[1])) × fact(ord(S[2])) × …… × fact(ord(S[N - 1]))

In other words, it is the product of the factorial of the **ord()** value of all the characters in **S**. (That's right, no modulus! Unlike the lame polynomial hashing.)

Not only that we have a new hashing mechanism in place, but we would also like to crack this now. Given a string S_1 consisting of lowercase letters only, your task is to find **a different** string S_2 consisting of lowercase letters**,** such that, **fake_hash(S_1) = fake_hash(S_2)** and $S_1 \neq S_2$.

If there are multiple possible choices for S_2**,** you need to find the **lexicographically smallest one,** or output the word **"Impossible"** without quotes, if it is not possible to find such a string.

Input

The first line contains an integer **T (1 ≤ T ≤ 3000),** denoting the number of test cases. Each test case contains the string S_1 **(1 ≤ $|S_1|$ ≤ 30)** consisting of lowercase letters only.

Output

For each test case, output the case number followed by the required output. Please refer to the sample input/output section for the precise format.

Constraints:

Except for the sample, the following constraints will hold:

1 ≤ $|S_1|$ ≤ 5, for **90%** of the test cases
1 ≤ $|S_1|$ ≤ 15, for **99%** of the test cases

Sample Input

```
10
tourist
petr
mnbvmar
bmerry
xellos
sevenkplus
dragoon
zzz
snapdragon
zosovoghisktwnopqrstuvwxyzoos
```

Output for Sample Input

```
Case 1: aaaaabbdnstttu
Case 2: aqst
Case 3: abmmnrv
Case 4: aaabbnrry
Case 5: aaaaaaadddlnuz
Case 6: aaaaaaabbddddnquuuz
Case 7: aaaaaaaaaaaaabdnnnt
Case 8: Impossible
Case 9: aaaaaaaaaabdnnnpst
Case 10: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaffffjnnnnqqttttuuuuxxzzzzzz
```

Problem E

Input: Standard Input
Output: Standard Output

Helping the HR

The University of **STU** has imposed some strict attendance rules for the faculty members. You are now being asked to write a program that will help the HR Department to monitor the monthly attendance records of its faculty members. The conditions are as follows:

1. A faculty member can be involved in either Day Shift or Evening Shift in a single day.
2. Faculty members involved in day shift classes in a day must come to the office by 9:30 AM and stay for at least 8 hours. However, hours spent in the office before 8:30 AM won't get counted.
3. Faculty members involved in the evening shift classes in a day must come to the office by 12:30 PM and must stay in the office for at least 9 hours. However, hours spent in the office before 8:30 AM won't get counted.
4. If a faculty member is late or fails to maintain required hours (8 or 9 hours based on involvement in day or evening shift classes) for more than three times in a month, he/she will be issued a show cause letter. Otherwise, 1 point will be deducted per late attendance or for failing to maintain required hours but no show cause letter will be issued.
5. If a faculty member is late and fails to maintain required hours in a single day, it will be counted as one point deduction. In other words maximum one point will be deducted for violation of more than one rule in a single day.

Given the number of days to be considered for attendance in a month, and the entry and exit time for each of those days for a faculty member, you will have to report one of the following about that employee:

1. "All OK"
2. "1 Point(s) Deducted"
3. "2 Point(s) Deducted"
4. "3 Point(s) Deducted"
5. "Issue Show Cause Letter"

Input

The input file contains several test cases.

The first line of each test case contains an integer **N (0<N<31)**, which denotes the total number of days in a month that needs to be considered. Each of the next **N** lines contains the description for a day. The description follows the following format:

`S:h1:m1:s1:h2:m2:s2`

Input is terminated by a case where **N=0**.

The value of **S** can be either 'D' or 'E' (Without the quotes), 'D' means that on this day the faculty member is involved in only day shift classes and 'E' means that the faculty member is involved in only evening shift classes. h1:m1:s1 denotes the entry time of the faculty member and h2:m2:s2 denotes the exit time in 24-hour format. You can assume that the times will be valid ($0 \le h_1, h_2 < 24$, $0 \le m_1, m_2, s_1, s_2 < 60$) and entry time will never be after exit time.

Output

For each set of input produce one line of output. This line contains any one of the lines "All OK", "1 Point(s) Deducted", "2 Point(s) Deducted", "3 Point(s) Deducted", "Issue Show Cause Letter" (Without the quotes)

Sample Input	Output for Sample Input
3 D:8:30:00:17:30:20 D:9:30:01:17:30:20 E:11:30:01:20:31:00 3 D:8:30:00:16:30:00 D:9:30:00:17:30:20 E:11:30:01:20:31:00 3 D:8:30:00:17:30:20 D:9:30:01:17:30:20 E:11:30:01:20:00:00 4 D:8:00:00:16:15:20 D:9:30:01:17:30:00 E:11:30:01:20:00:00 E:11:30:01:20:00:00 0	1 Point(s) Deducted All OK 2 Point(s) Deducted Issue Show Cause Letter

Problem F

Input: Standard Input
Output: Standard Output

Path Intersection

Do you know trees? They have branches and leaves. Their roots stay inside the ground to make a solid base and draw water from the soil. People get oxygen and foods from them.

In graph theory **Tree** is known as a connected graph consisting of **N** nodes and **N-1** edges. The nodes are denoted by integers from **1** to **N**. So there is exactly one path between any two nodes. The root stays on top making the whole tree hang under it. People ask silly and nerdy questions about it. Speaking of one such silly questions, do you happen to know how to find out the number of common nodes among all the **K** given paths in a tree?

Input

There will be **T (1 ≤ T ≤ 50)** test cases. Each case will start with an integer **N (1 ≤ N ≤ 10000),** representing the number of nodes in the tree. Then, **N-1** lines will follow, each having two integers **u** and **v (1 ≤ u, v ≤ N),** representing an undirected edge from **u** to **v**. The next line will contain an integer **Q (1 ≤ Q ≤ 10000),** denoting the number of queries. Each query will start with an integer **K (2 ≤ K ≤ 50)** denoting the number of paths. Then, **K** lines will follow each having two integers **a** and **b (1 ≤ a, b ≤ N),** representing a path from **a** to **b**. For the entire input file, **∑N ≤ 500000, ∑Q ≤ 500000, ∑K ≤ 500000**. The dataset is huge, so use faster I/O methods.

Output

For each test case, print the case number in a line. Then for each query, print an integer in a line denoting the number of common nodes among all the **K** given paths.

Sample Input

```
2
6
1 2
2 3
3 4
2 5
2 6
2
2
4 5
3 6
3
4 5
3 1
2 6
2
1 2
1
2
1 1
2 2
```

Output for Sample Input

```
Case 1:
2
1
Case 2:
0
```

Problem G
Techland

Input: Standard Input
Output: Standard Output

Rahim is very much enthusiastic about technologies. He is always ready to buy new gadgets and stuff. But before buying anything, he always watches a lot of reviews and gathers information about that specific stuff. The country **"TechLand"** where Rahim lives, doesn't believe in a competitive market. That's why all the products in TechLand are of the same price. When Rahim wants to buy anything, he visits the closest shop from his position and buys the product.

TechLand has **N** cities which are connected with **N-1** bi-directional roads. The cities are denoted by integers from **1** to **N**. In this country, sometimes new companies come to do some business and sometimes they leave the country. So, the shop from where Rahim will buy his gadgets is not always fixed.

In this situation, you will be given **Q** events. The events can be one of the following:

1. **X L R:** The company **X** is going to start new business and they will open shops at the cities labeled with **L, L+1, L+2, …, R-2, R-1, R (L ≤ R).** If the company already exists in the country, then they will shut down all the shops opened previously and will open new shops in these cities (whose labels are between **L and R inclusive**). In a single city, there can be multiple shops, but no company will have multiple shops in a city.
2. **X**: Company **X** is leaving TechLand. It is guaranteed that the company **X** was doing their business before leaving the country. In the future, the company may or may not come back again.
3. **C M P_1 P_2 … P_M**: Rahim wants to buy a new gadget. Currently, he is at city **C**. He has also prepared a preferred list of companies for purchasing the gadget. The list contains **M** companies **P_1, P_2, P_3, … , P_M**. To buy his gadget, Rahim will travel to the closest city from **C** where at least one of the preferred companies has a shop. It is guaranteed that each of the companies in Rahim's preferred list has come at least once in the country to establish their business. But it is not guaranteed that they are still continuing their business. Sometimes it may happen that Rahim is unable to find any shop of those **M** companies. In that situation, you will also need to report that.

Rahim is not that good at finding the closest city he needs to travel. You guys are good at programming! So help him to figure out the shortest distance he will need to travel to buy the gadget for each of the **type 3** events.

Input

The first line of the input will contain a single integer **T (1 ≤ T ≤ 10)** denoting the number of test cases. Each test case contains several lines. The first line of each test case will contain an integer **N (1 ≤ N ≤ 50000)** which denotes the number of the cities in TechLand. Each of the next **N-1** lines contains two integers **U V (1 ≤ U, V ≤ N and U≠V)** which means there is a bidirectional road connecting cities **U** and **V**.

After that the next line will have a single integer **Q (1 ≤ Q ≤ 100000)** representing the number of events. Each of the next **Q** lines describes a single event. Each event line starts with an integer **E (1 ≤ E ≤ 3)** denoting the event type.

If **E = 1**, then it will contain three more integers **X, L,** and **R (1 ≤ X ≤ 100000** and **1 ≤ L ≤ R ≤ N)**.
If **E = 2,** then the line will contain another integer **X (1 ≤ X ≤ 100000)**.

If **E = 3**, the line will start with two integers **C** and **M (1 ≤ C ≤ N** and **1 ≤ M ≤ 100000)**. After these two integers there will be **M** more integers $P_1, P_2, P_3, \ldots, P_M$ **(1 ≤ P_i ≤ 100000)**.

Each event is described elaborately in the description above. In a single test case the sum of **M** over all the events will not exceed **100000**.

Note: Input file is huge. Please use faster I/O.

Output

The first line of each test case should contain the case number in the format: **"Case Y:**" without quotes, where **Y** denotes the test case number. After that for each event of **type 3**, a single line should contain the minimum distance Rahim will need to travel in order to buy the gadget. If it is not possible to find such a city where Rahim should travel then just output **-1** instead.

Sample Input	Output for Sample Input
2 6 1 4 1 5 1 6 5 2 5 3 4 1 1 2 3 3 5 1 1 2 1 3 4 1 1 2 1 2 2 1 100000 1 1 3 2 1 100000	Case 1: 1 -1 Case 2: 1

Problem H

Input: Standard Input
Output: Standard Output

Tile Game

We have a board and there are some square shaped tiles inside it. The board and the tiles look similar to the picture on the right:

As you can see the board has some borders, and the tiles inside the board are solid square shaped. If you tilt the board in some direction, all the tiles will try moving to that direction until they hit the board boundary or some other tiles. We can tilt the board in four directions: up, down, left and right. For example, let's consider following 3×3 board with 4 tiles inside. The tiles are marked from 1 to 4:

Initial state

		3
1	4	
	2	

For your convenience, we have shown below the resulting board state after tilting the initial board in each of the four directions.

upward	1 4 3 2	leftward	3 1 4 2
downward	4 1 2 3	rightward	3 1 4 2

We call a board state **stable** if the tiles don't move if we tilt it leftward or downward. That is, the tiles will be cluttering around the bottom-left corner. You are given an (R × C) shaped **stable** board (R rows and C columns). There are some tiles inside it and each of these tiles is marked with **distinct colors**. You can tilt the board in any direction any number of times. You need to find the number of distinct stable board states that you can arrive at.

Two stable boards are distinct if in one board we have a color at one cell but in the other board, we have the same color in some different cell.

Input

In the first line of the input, you are given the number of test cases, **T (1 ≤ T ≤ 100)**. Hence follows T test cases.

Each test case starts with two positive integers **R** and **C (1 ≤ R, C ≤ 200)**. Then following R lines contain C characters each, denoting an R×C shaped **stable** board. Each character is either a dot (.) or a hash (#). A hash denotes a tile. It's guaranteed that there is at least one tile in the board.

Output

For each test case, print the case number followed by the answer of the test case. Since the answer can be large, print modulo **78294349** of it. For the specific output format, please consult the sample input/output.

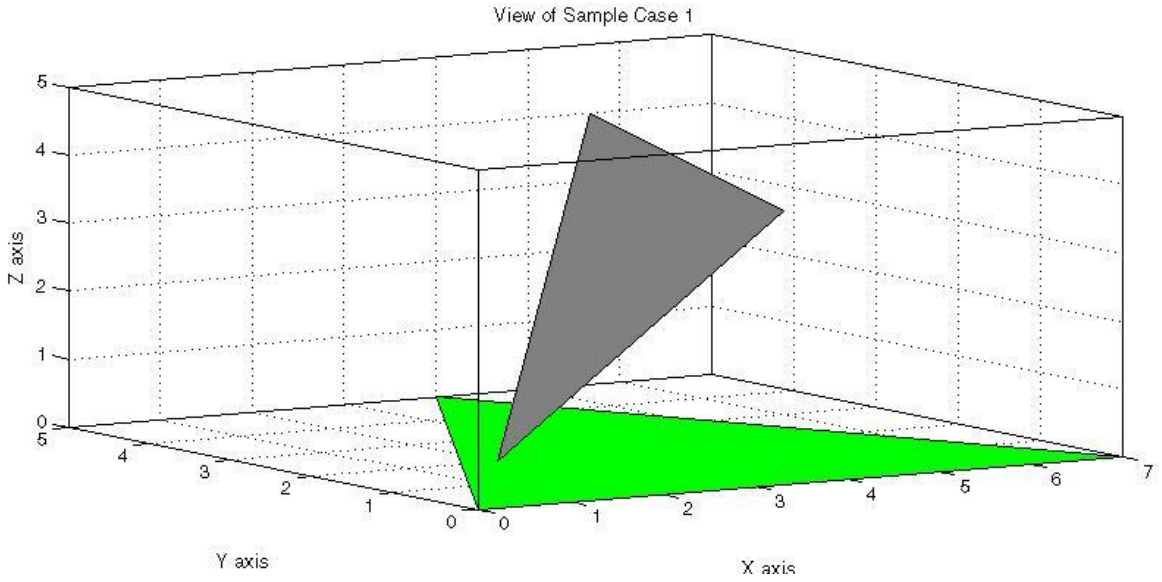
Sample Input	Output for Sample Input
3 2 2 .. #. 2 2 #. #. 18 13 ..... ##..... ##..... ####..... ####..... #####..... #####..... #######..... #######..... #########.... #########.... ##########... ###########.. ###########.. ###########.. ############. ############# #############	Case 1: 1 Case 2: 1 Case 3: 27457584

Problem I

Input: Standard Input
Output: Standard Output

Triangles

In this problem, you are given the vertices of two triangles in **3D** space. Determine the minimum distance between the plane segments bounded by the edges of the triangles. See the image below and samples for a clear understanding.



Input

The first line will contain a positive integer **T (1 ≤ T ≤ 10000)**, the number of test cases.

For each test case, two lines will contain **9** space separated integers each, denoting **3** coordinates of the vertices of the two triangles. The triangles will have a **non-zero** area. The absolute value of the coordinates will not exceed 10^5.

Consecutive test cases will be separated by a new line.

Output

Print the distance. Absolute error less than 10^{-6} will be ignored. Please check the sample for details.

Sample Input	Output for Sample Input
2 0 0 0 7 0 0 4 5 0 2 2 0 3 2 5 6 3 3 0 0 0 7 0 0 4 5 0 2 2 1 5 4 10 5 4 11	Case 1: 0.000000000 Case 2: 1.000000000

Problem J

Input: Standard Input
Output: Standard Output

VAT Man

The people of Wakanda pay income tax every year to fund all the initiatives of the Government. To facilitate all the newly introduced radical development plans however, the Government needs a lot more fund. On top of the regular income tax, the citizens of Wakanda are now required to pay an additional 15% on any purchase. This extra amount is going to be collected as VAT (Vibranium Acquisition Tax). Wakanda needs your help to calculate the price of any product; Wakanda wants you to be their VAT Man!

Input

The input consists of several test cases. The first line gives you the number of test cases, **T (T < 100)**. For each of the next **T** lines, you'll get exactly one integer -- the price of the product, **P (1 ≤ P ≤ 10000)**.

Output

For each test case, you only need to print the total price of the product including 15% VAT. Print your answer rounded to two decimal places.

The following table shows you code snippets in C, C++, and Java to calculate the total price (including 15% VAT) of a product that's worth 100 WD (Wakandian Dollar). It also shows you how to print your answer rounded to two decimal places.

C	C++	Java

#include <stdio.h>
…
 	
double price = 100;
double output = price * 1.15;
printf("%.2lf\n", output);
	#include <iostream>
	#include <iomanip>
	using namespace std;
	…
	
	double price = 100;
	double output = price * 1.15;
	cout << fixed << setprecision(2)
	  << output << endl;
		double price = 100;
		double output = price * 1.15;
		System.out.printf("%.2lf\n",
		  output);

Sample Input

```
2
100
1021
```

Output for Sample Input

```
115.00
1174.15
```