A Maximize Engine

Maximize Engine is a machine that takes a hexadecimal integer number and makes the maximum hexadecimal number out of it using all the digit in that number. The machine works on step by step. At every step the machine picks up a digit and either move it to the beginning of the number or at the end of the number.

For example, if you give 43251 to the engine, it will give you 54321, the maximum number can produce from 14325 by moving 5 to the front in one step.
Another example, for 132, the machine will give us 321, by moving 1 to the end the number in one step.

But the problem in the machine is, it doesn't work optimally. Sometimes it uses more steps than the optimal one to make a maximum number.

Now here is your task, you have to simulate the Maximize Engine. You are given a hexadecimal integer number, you have to find out the minimum step to make the number maximum using the Maximize Engine rule.

Input:

First line of the input is an integer ***T*** (1 <= ***T***< 200), the number of test cases. Each of the following ***T*** lines contain a hexadecimal number ***N***, which is no longer than 16 digits consisting of 0-9 and A-F.

Output:

For each test case, print the minimum number of steps required to get the maximum number.

Example:

Sample Input	Sample Output
3 43215 51254 ABF	1 2 2

B Mr. Upo

Mr. Upo is a great programmer and proud owner of a very popular mobile application. Everyday many users use and rate the application. Each user can rate the application with a value **0 to 5**. But unfortunately app store updates **Application Rating** everyday only once and they do not show how many users have rated the application. Application Rating is the average of the ratings given by users.

Mr. Upo desperately wants to know the number of users that rated his application. After some thinking he realized, it's not possible for him to know the number of users. Now, Mr. Upo provides you the daily ratings of his application and wants you to calculate the least number of users that must have rated his application.

Input:

First line of input is the number of Test cases **T (T <= 100)**. Each test case starts with a number **N (1 <= N <= 10^3)**, the number of daily ratings. Next line contains **N** space separated decimal numbers. Decimal portion of every number is exactly 3 place. After reducing 3 places, decimals like 0.9995 or 0.9996 becomes 1.000, but 0.99949 becomes 0.999. See sample input for examples.

Output:

For each test case output case number and the least number of users that must have rated the application. Output for all input dataset is less than 2^{60}. See sample output for example.

Example:

Sample Input	Sample Output
2 1 1.000 2 1.500 1.300	Case 1: 1 Case 2: 10

C Corrupted and Uncorrupted

Landx is a country that is always face natural disasters. During the rainy season, heavy rains coupled with cyclones create widespread damage to life and property in some parts of the country.

As such, the proper distribution and management of aid and other materials becomes first priority for the government to ensure the well-being of the affected people.

To make the task of distribution easier, the government has decided on a strategy. It has decided to give the aid in the hands of a few representatives from a community, who will then distribute all the aid to people within their community.

Unfortunately, this is hampered by the presence of unscrupulous people present in each area. These people divide the aid among themselves and deprive the other citizens of their share. The government does not want to give the aid in the hands of these corrupt people. The respective minister worked very hard and racked his brains all night long and could not come up with a solution of this problem. So he decided to hire some programmers to do the job for him. He explains the problem to the programs with the following statements.

If a person A can give aid to a person B and person B can give aid to a person C, then we can say, person A can "give to" person C. If a person A can give aid to a person B, then we can say, B can "receive from" A. A set of people among a community is called a "corrupt group" if all of the followings are true. All the persons who are not in the member of any "corrupted group", they are within the "uncorrupted people".

(i) There are at least 2 people in the set
(ii) Any person in the set can "give to" any other person
(iii) There is no other people out of the set, who can both "give to" and "receive from" the group and

Your task is to find all the corrupted groups in each community. Also find out the "uncorrupted people" that would be deprived if the aid went into the hands of the corrupted group.

Input:

First input line contains an integer T (T<150) denotes the number of input sets. First line of each input set contains two numbers N and M (1<=N<=10000, 0<=M<=100000).

Here, N is the number of people in a community and M is the number of relation among them. In next M lines there will be two integer X & Y (1<=X, Y<=N &&X != Y) indicating X gets aid then he will share it (give some part of it) with Y.

Output:

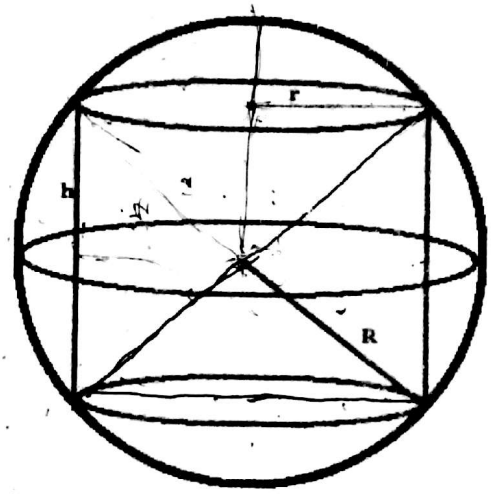
For each test case, output "Case #C:", where C represents the number of test case (starting from 1). Then next lines will contain the group member lists, one line for each group. Then print a line of the list of uncorrupted people. Each group member list will be sorted and groups will be sorted according to their first member. List of uncorrupted people also will be sorted. See the sample output for the output format.

Example:

Sample Input	Sample Output
3 5 4 1 2 2 1 3 4 4 3 3 0 4 4 1 2 2 1 3 4 4 3	Case #1: Corrupted Group 1: 1 2 Corrupted Group 2: 3 4 Uncorrupted People: 5 Case #2: Uncorrupted People: 1 2 3 Case #3: Corrupted Group 1: 1 2 Corrupted Group 2: 3 4

D Cylinder

You are given a sphere of radius R. Find the largest cylinder volume that can be fit inside the sphere.



Input:

First line in input is T (at most 1000), the number of test cases. Then T lines follow, each containing an integer R (1 <= R <= 10000), the radius of the sphere.

Output:

For each test case, print the case number starting from 1, and the maximum possible volume of the cylinder rounded off to 3 decimal places. Read sample input and output section for details. Floating point error lower than 1e-3 will be ignored by judge.

Example:

Sample Input	Sample Output
3 2 6 45	Case 1: 19.347 Case 2: 522.374 Case 3: 220376.623

G Minimum Cuts

For a given string, you can cut the string into pieces, so that each single piece is a palindromic string. For example, look at the following string "abbaabbacbab", you can cut it as shown below:

abba | abba | c | bab

So, with 3 cuts, you can get all the palindromes separated. However, you can do better, you can achieve the same goal with one less cut:

abbaabba | c | bab

In this problem, you will have to find the minimum number of cuts needed to sort out all the palindromic pieces find out all the palindromes within the string. Note, all the pieces have to be disjoint, and when concatenated together in the order of occurrence, they must form the given string.

Input:

First line of the input contains an integer T (T <= 100), the number of test cases. For each case, there will be one line of input, a string S consisting of only lowercase ASCII letters. Here, S is from 1 to 100 characters long.

Output:

For each test case, print the test case number starting from 1 and then the desired output. See sample input and output for details.

Example:

Sample Input	Sample Output
6 a ab aba abcb abbaab ababba	Case 1: 0 Case 2: 1 Case 3: 0 Case 4: 1 Case 5: 2 Case 6: 2

H Mastering Shopping

After lots of hard work, you are now able to solve all the common dynamic programming problems pretty easily. Now to test your skills, your trainer changed the common dynamic programming problems a bit and it's upon you to solve this problem [with or without dynamic programming is up to you (:].

There will be a list of items for you to buy and you have to buy each item only once. These items can be found at multiple stores (or shops) and their price may vary from store to store. From each store, you have to buy at-least one item and cannot buy more than X items.

You have to determine the minimum total cost with which you can buy all the items on the list.

Input:

Input starts with the number of test cases, T (T <= 20).

Each test case begins with two integers, n (0 < n <= 100) and m (0 < m <= 100). Where n denotes the number of items you have to buy and m denotes the number of shops you can buy from. Next line will contain m integers Xi (1 <= Xi <= 100), each denoting the maximum number of items you can buy from the i-th shop.

Next line will contain an integer r (1 <= r <= 1,000). Each of the next r lines contains 3 space separated integers u (1 <= u <= n), v (1 <= v <= m) and w (1 <= w <= 10,000), denoting that the u-th item can be found at the v-th shop and will cost w units.

Output:

For each case, print the case number and the minimum cost to buy all the items or "impossible" if that is not possible.

Example:

Sample Input	Sample Output
2 2 2 2 1 3 1 1 5 2 1 5 2 2 9 3 2 2 1 3 1 1 5 2 1 5 2 2 9	Case 1: 14 Case 2: impossible

I Wythoff's Game

Wythoff's game is a two player mathematical game played with two piles of stones. Players take turn and remove stones from one or both piles.

Alice and Bob were playing of Wythoff's game with two piles of stones and then got bored. Then they decided to increase the number of piles to three. In the version of their game, there will be three piles of stone. Each player takes stones of the piles in turn and the player who takes the last stones wins. In each turn, a player can take N stones (at least one) from one pile or two piles or all three piles. When taking stone from multiple piles in a move, N must be equal for the piles from where the player chooses to remove stones.

For example, from a pile state of (8, 10, 15), a move to (8, 0, 15) is a valid move. (10 stones from 2nd pile). From a pile state of (8, 10, 15), a move to (8, 5, 10) is a valid move. (5 stones from 2nd & 3rd pile). From a pile state of (8, 10, 15), a move to (5, 7, 12) is a valid move. (3 stones from each pile) and so on. But from a pile state of (8, 10, 15), a move to (8, 7, 10) is an invalid move. (3 from 2nd pile and 5 from 3rd pile). Number of stones must be equal when removing from multiple piles.

Alice will start the game. For a given state of the piles, your task is to determine who will win if Alice and Bob both play optimally.

Input:

Input starts with an integer **T (≤ 200)**, denoting the number of test cases. For each test case there will be a line containing 3 integers **(0 ≤ Xi, Yi, Zi ≤ 100)** denoting number of stones in each pile.

Output:

For each case, print the case number and **'Alice wins!'** or **'Bob wins!'** without quotes depending on the result. See sample input output for more info.

Example:

Sample Input	Sample Output
3 0 0 5 1 5 5 5 5 5	Case #1: Alice wins! Case #2: Bob wins! Case #3: Alice wins!