AGRA - Árboles y Grafos, 2023-2
Tarea 2: Semanas 9, 10, 11, 12

Para entregar el domingo 22 de octubre de 2023

Problemas prácticos a las 23:59 en la arena de programación

Tanto los ejercicios como los problemas deben ser resueltos, pero únicamente las soluciones de los problemas deben ser entregadas. La intención de los ejercicios es entrenarlo para que domine el material del curso; a pesar de que no debe entregar soluciones a los ejercicios, usted es responsable del material cubierto en ellos.

Instrucciones para la entrega

Para esta tarea y todas las tareas futuras, la entrega de soluciones es *individual*. Por favor escriba claramente su nombre, código de estudiante y sección en cada hoja impresa entregada o en cada archivo de código (a modo de comentario). Adicionalmente, agregue la información de fecha y nombres de compañeros con los que colaboró; igualmente cite cualquier fuente de información que utilizó.

¿Cómo describir un algoritmo?

En algunos ejercicios y problemas se pide "dar un algoritmo" para resolver un problema. Una solución debe tomar la forma de un pequeño ensayo (es decir, un par de párrafos). En particular, una solución debe resumir en un párrafo el problema y cuáles son los resultados de la solución. Además, se deben incluir párrafos con la siguiente información:

- una descripción del algoritmo en castellano y, si es útil, pseudo-código;
- por lo menos un diagrama o ejemplo que muestre cómo funciona el algoritmo;
- una demostración de la corrección del algoritmo; y
- un análisis de la complejidad temporal del algoritmo.

Recuerde que su objetivo es comunicar claramente un algoritmo. Las soluciones algorítmicas correctas y descritas *claramente* recibirán alta calificación; soluciones complejas, obtusas o mal presentadas recibirán baja calificación.

Hay seis problemas prácticos cuyos enunciados aparecen a partir de la siguiente página.

A - Avoiding Your Boss

Source file name: `boss.py`
Time limit: 5 seconds

Imagine that like me you are also an idle but reasonably good programmer. On one fine morning you wake up from sleep and discover that your bed is such a lovely place and you must not leave it for the miserable place called office. So you just pick up your mobile phone, call your boss with a gloomy voice and tell him that you are quite sick. Your boss is a very strict but sympathetic person and so he grants you leave via phone. As the day grows old you discover that you must go out for shopping. But you are afraid that your boss might see you. So you need to know if there is any safe option to reach the market. You know that your boss stays only in two places, his residence and the office. While going from one place to another he uses a path that does not cost more than any other path. So you must avoid such paths and even places that your boss may reach. Your boss must not see you outside your home. You can assume that the cost of reaching another location in the same place is zero and you must go outside your home to reach market and your boss must come outside to reach home (from office) or office (from home). As your job is at stake so you cannot take any chances.

Input

The input file contains several sets of input. Each set of input starts with six integers P, R, BH, OF, YH and M. Here

P = Total number of places. $(0 < P \leq 100)$

R = Total number of connecting roads. $(0 \leq R \leq 4950)$

BH = The Place where your boss lives. $(0 < BH \leq P)$

OF = The place where your office is situated. $(0 < OF \leq P)$

YH = The place where your home is situated. $(0 < YH \leq P)$

M = The place where the market is situated. $(0 < M \leq P)$

Next R lines contain description of the city. Each line contains three integers p_1, p_2 and d. These three integers denote that place p_1 and place p_2 is connected by a road whose cost is d. Here $(0 < p_1, p_2 \leq P)$ and d is a positive integer less than 101. Streets are all bi-directional. You can assume that two different places are connected by only one road. Input is terminated by end of file.

The input must be read from standard input.

Output

For each set of input produce one line of output. This line contains the cost of going from your home to the market. If it is not possible for you to go to the market avoiding your boss print the line `'MISSION IMPOSSIBLE.'` (without the quotes).

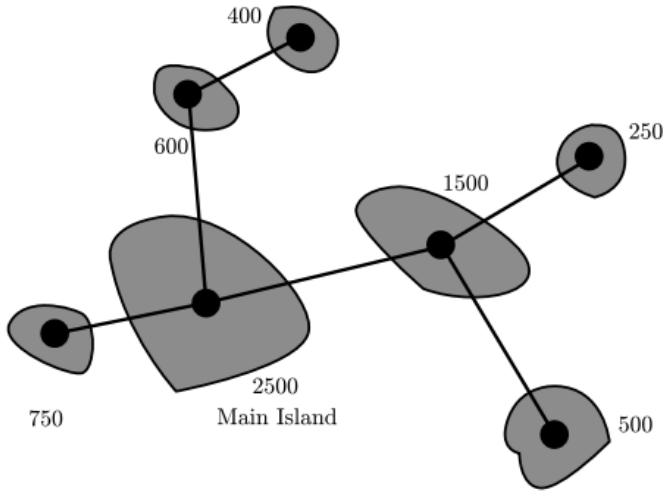
The output must be written to standard output.

Sample Input	**Sample Output**
3 2 2 3 1 3 1 2 4 2 3 4 3 2 2 3 3 3 1 2 4 2 3 4 4 3 2 3 1 4 1 2 4 2 3 4 1 4 10	MISSION IMPOSSIBLE. MISSION IMPOSSIBLE. 10

B - Island Hopping

Source file name: `island.py`
Time limit: 1 second

The company Pacific Island Net (PIN) has identified several small island groups in the Pacific that do not have a fast internet connection. PIN plans to tap this potential market by offering internet service to the island inhabitants. Each groups of islands already has a deep-sea cable that connects the main island to the closest internet hub on the mainland (be it America, Australia or Asia). All that remains to be done is to connect the islands in a group to each other. You must write a program to help them determine a connection procedure.



For each island, you are given the position of its router and the number of island inhabitants. In the figure, the dark dots are the routers and the numbers are the numbers of inhabitants. PIN will build connections between pairs of routers such that every router has a path to the main island. PIN has decided to build the network such that the total amount of cable used is minimal. Under this restriction, there may be several optimal networks. However, it does not matter to PIN which of the optimal networks is built.

PIN is interested in the average time required for new customers to access the internet, based on the assumption that construction on all cable links in the network begins at the same time. Cable links can be constructed at a rate of one kilometer of cable per day. As a result, shorter cable links are completed before the longer links. An island will have internet access as soon as there is a path from the island to the main island along completed cable links. If m_i is the number of inhabitants of the i^{th} island and t_i is the time when the island is connected to the internet, then the average connection time is:

$$\frac{\sum t_i * m_i}{\sum m_i}$$

Input

The input consists of several descriptions of groups of islands. The first line of each description contains a single positive integer n, the number of islands in the group ($n \leq 50$). Each of the next n lines has three integers x_i, y_i, m_i, giving the position of the router (x_i, y_i) and number of inhabitants m_i ($m_i > 0$) of the islands. Coordinates are measured in kilometers. The first island in this sequence is the main island.

The input is terminated by the number zero on a line by itself.

The input must be read from standard input.

Output

For each group of islands in the input, output the sequence number of the group and the average number of days until the inhabitants are connected to the internet. The number of days should have two digits to the right of the decimal point. Use the output format in the sample given below.

Place a blank line after the output of each test case.

The output must be written to standard output.

Sample Input	**Sample Output**
7 11 12 2500 14 17 1500 9 9 750 7 15 600 19 16 500 8 18 400 15 21 250 0	Island Group: 1 Average 3.20

C - Anti Brute Force Lock

Source file name: `lock.py`
Time limit: 6 seconds

Lately, there is one serious problem with Panda Land Safe Box: several safes have been robbed! The safes are using old 4-digits rolling lock combination (you only have to roll the digit, either up or down, until all four of them match the key). Each digit is designed to roll from 0 to 9. Rolling-up at 9 will make the digit become 0, and rolling-down at 0 will make the digit become 9. Since there are only 10000 possible keys, 0000 through 9999, anyone can try all the combinations until the safe is unlocked.

What's done is done. But in order to slow down future robbers' attack, Panda Security Agency (PSA) has devised a new safer lock with multiple keys. Instead of using only one key combination as the key, the lock now can have up to N keys which has to be all unlocked before the safe can be opened. These locks works as the following:

- Initially the digits are at 0000.
- Keys can be unlocked in any order, by setting the digits in the lock to match the desired key and then pressing the UNLOCK button.
- A magic JUMP button can turn the digits into any of the unlocked keys without doing any rolling.
- The safe will be unlocked if and only if all the keys are unlocked in a minimum total amount of rolling, excluding JUMP (yes, this feature is the coolest one).
- If the number of rolling is exceeded, then the digits will be reset to 0000 and all the keys will be locked again. In other word, the state of the lock will be reset the cracking is failed.

PSA is quite confident that this new system will slow down the cracking, giving them enough time to identify and catch the robbers. In order to determine the minimum number of rolling needed, PSA wants you to write a program. Given all the keys, calculate the minimum number of rolls needed to unlock the safe.

Input

The first line of input contains an integer T, the number of test cases follow. Each case begins with an integer N ($1 \leq N \leq 500$), the number of keys. The next N lines, each contains exactly four digits number (leading zero allowed) representing the keys to be unlocked.

The input must be read from standard input.

Output

For each case, print in a single line the minimum number of rolls needed to unlock all the keys.

Explanation for the 2nd case:

- Turn `0000` into `1111`, rolls: 4
- Turn `1111` into `1155`, rolls: 8
- Jump `1155` into `1111`, we can do this because `1111` has been unlocked before.
- Turn `1111` into `5511` rolls: 8

  Total rolls = 4 + 8 + 8 = 20

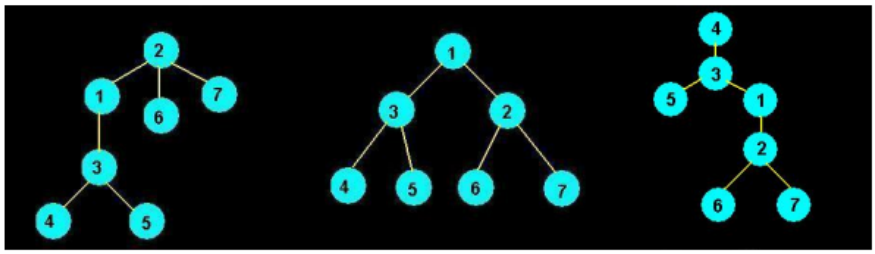
The output must be written to standard output.

Sample Input	**Sample Output**
4 2 1155 2211 3 1111 1155 5511 3 1234 5678 9090 4 2145 0213 9113 8113	16 20 26 17

D - The Tree Root

Source file name: `root.py`
Time limit: 1 second

Tree is an important data structure. Searching is a basic operation in any data structure. In a tree searching mainly depends on its height. Consider the following three trees.



If you observe carefully, you will see that all trees are same except different nodes are used as roots. Here the height of the tree varies with the selection of the root. In the 1st tree root is `'2'` and height is 3. In 2nd one root is `'1'` and height is 2. And in last one root is `'4'` and height is 4. We will call `'1'` best root as it keeps the tree with the least possible height and `'4'` worst root for the opposite reason.

In this problem, you have to find out all best roots and worst roots for a given tree.

Input

Each dataset starts with a positive integer N ($3 \leq N \leq 5000$), which is the number of nodes in the tree. Each node in the tree has a unique id from 1 to N. Then successively for each i' th node there will be a positive integer $K[i]$ following id of $K[i]$ nodes which are adjacent to i. Input is terminated by EOF.

The input must be read from standard input.

Output

For each dataset print two lines. In the 1st line show all the best roots in ascending order and in next line show all worst roots in ascending order. See sample output for exact format.

The output must be written to standard output.

Sample Input	**Sample Output**

`7`
`2 2 3`
`3 1 6 7`
`3 1 4 5`
`1 3`
`1 3`
`1 2`
`1 2`
	`Best Roots  : 1`
	`Worst Roots : 4 5 6 7`

E - RMQ with Shifts

Source file name: `shifts.py`
Time limit: 2 seconds

In the traditional RMQ (Range Minimum Query) problem, we have a static array A. Then for each query (L, R) $(L \leq R)$, we report the minimum value among $A[L], A[L+1], \ldots, A[R]$. Note that the indices start from 1, i.e. the left-most element is $A[1]$.

In this problem, the array A is no longer static: we need to support another operation

$$shift(i_1, i_2, i_3, \ldots, i_k)\ (i_1 < i_2 < \ldots < i_k, k > 1)$$

we do a left "circular shift" of $A[i_1], A[i_2], \ldots, A[i_k]$.

For example, if $A = \{6, \mathbf{2}, 4, \mathbf{8}, \mathbf{5}, 1, \mathbf{4}\}$, then $shift(2, 4, 5, 7)$ yields $\{\mathbf{6}, \mathbf{8}, 4, 5, 4, 1, 2\}$. After that, $shift(1, 2)$ yields $8, 6, 4, 5, 4, 1, 2$.

Input

There will be only one test case, beginning with two integers n, q $(1 \leq n \leq 100000,\ 1 \leq q \leq 250000)$, the number of integers in array A, and the number of operations. The next line contains n positive integers not greater than 100000, the initial elements in array A. Each of the next q lines contains an operation. Each operation is formatted as a string having no more than 30 characters, with no space characters inside. All operations are guaranteed to be valid.

Warning: The dataset is large, better to use faster I/O methods.

The input must be read from standard input.

Output

For each query, print the minimum value (rather than index) in the requested range.

The output must be written to standard output.

Sample Input	**Sample Output**

`7 5`
`6 2 4 8 5 1 4`
`query(3,7)`
`shift(2,4,5,7)`
`query(1,4)`
`shift(1,2)`
`query(2,2)`
	`1`
	`4`
	`6`

F - Fire Station

Source file name: `station.py`
Time limit: 2 seconds

A city is served by a number of fire stations. Some residents have complained that the distance from their houses to the nearest station is too far, so a new station is to be built. You are to choose the location of the fire station so as to reduce the distance to the nearest station from the houses of the disgruntled residents.

The city has up to 500 intersections, connected by road segments of various lengths. No more than 20 road segments intersect at a given intersection. The location of houses and firestations alike are considered to be at intersections (the travel distance from the intersection to the actual building can be discounted). Furthermore, we assume that there is at least one house associated with every intersection. There may be more than one firestation per intersection.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The first line of input contains two positive integers: f, the number of existing fire stations ($f \leq 100$) and i, the number of intersections ($i \leq 500$). The intersections are numbered from 1 to i consecutively. f lines follow; each contains the intersection number at which an existing fire station is found. A number of lines follow, each containing three positive integers: the number of an intersection, the number of a different intersection, and the length of the road segment connecting the intersections. All road segments are two-way (at least as far as fire engines are concerned), and there will exist a route between any pair of intersections.

The input must be read from standard input.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

You are to output a single integer: the lowest intersection number at which a new fire station should be built so as to minimize the maximum distance from any intersection to the nearest fire station.

The output must be written to standard output.

Sample Input	**Sample Output**
1 1 6 2 1 2 10 2 3 10 3 4 10 4 5 10 5 6 10 6 1 10	5