ADA - Análisis y Diseño de Algoritmos, 2025-1
Tarea 2: Semanas 4 y 5

Para entregar el domingo 23 de febrero de 2025

Problemas conceptuales a las 23:59 por BrightSpace

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.

Ejercicios

La siguiente colección de ejercicios, tomados del libro de Cormen et al. es para repasar y afianzar conceptos, pero no deben ser entregados como parte de la tarea.

14.1-2, 14.1-3, 14.2-1, 14.2-4, 14.4-1, 14.4-2, 14.4-5, 14.5-1.

Problemas conceptuales

1. Ejercicio 6.2: *Expert Computer Hackers* (Kleinberg & Tardos, página 313).
2. Ejercicio 6.8: *The City of Zion* (Kleinberg & Tardos, página 319).

Problemas prácticos

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

A - Arctic Virus

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

The year is 2045 and climate change has reached an irreversible tipping point. The polar ice caps are melting at an alarming rate, causing global sea levels to rise and destabilizing ecosystems. Scientists around the world have been racing to find solutions to slow the thawing of the poles. Then, one day, they discover something extraordinary hidden deep within the ice –a network of ancient, highly advanced technology. These dormant machines, left behind by an ancient civilization, have the power to control the Earth's temperature.

Reactivating this technology requires solving a series of complex computational problems to restore balance to the planet's climate. Ultimately, these problems reduce to finding specific instances of an ancient virus that can unlock the machines. Unfortunately, this virus no longer "walks" the Earth, so advanced in-vivo laboratory experimentation is required to recreate instances of the arctic virus that can trigger the unlocking process.

The virus consists of a chain of bases –adenine (A), cytosine (C), guanine (G), and thymine (T)– and evolves in one of the following forms:

$$\varphi ::= A \mid T \mid \varphi C \mid A\varphi \mid A\varphi^{-1} \mid G\varphi^{-1}C.$$

In its *simple stage*, the arctic virus takes one of two configurations: A and T. Any other form is considered a *mutation*, which can occur in four ways:

- The notation φC means that the virus φ mutates by adding C to the end of its chain of bases.
- The notation $A\varphi$ means that the virus φ mutates by appending A to the beginning its chain of bases.
- The notation $A\varphi^{-1}$ means that the virus φ mutates by appending A to the beginning of its reversed chain of bases.
- The notation $G\varphi^{-1}C$ means that the virus φ mutates by appending G to the beginning and C to the end of its reversed chain of bases.

For example, A, AT, $GTAC$, and $ACGTCCGA$ are configurations of the virus, while TTG, $CGGAT$, and $GAGAT$ are not.

Your task is to help the scientists unlock the secrets of this ancient technology by writing code to identify configurations of the arctic virus. In particular, you are tasked with designing and implementing an algorithm to detect if a chain of bases created in the laboratory corresponds to the virus, in any of its configurations, or not. This will allow the scientists to verify the results of in-vivo experimentation as quickly and efficiently as possible. Failure means the complete melting of the poles and catastrophic consequences for humanity.

Input

The input consists of several test cases. Each testcase comprises exactly one line of the form n s, with $1 \leq n \leq 1\,000$, and s, with $|s| = n$, a sequence comprising only the characters `A`, `C`, `G`, and `T` representing the four bases that can appear in the virus.

The input must be read from standard input.

Output

For each testcase, output a single line with:

- '`simple`' if s represents a simple stage of the virus;
- '`mutation`' if s represents a mutation of the virus; and
- '`doomed`' if s is not an instance of the arctic virus.

The output must be written to standard output.

Sample Input	**Sample Output**
1 A	simple
2 AT	mutation
4 GTAC	mutation
8 ACGTCCGA	mutation
3 TTG	doomed
5 CGGAT	doomed
5 GAGAT	doomed

B - Batman

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

Batman, the superhero, has difficult days fighting villains, and today is one of these days. Batman begins the day checking the list of superpowers that he can use to beat the enemies of Gotham City, and the list of villains that he has to beat during the day.

Each superpower has a name, an attack factor and an energy consumption in calories. Each villain has an alias, a defense factor, and the list of the names of the superpowers that can affect him. To beat a specific villain, Batman must attack him with a superpower that can affect the villain and whose attack factor is greater or equal than the defense factor of the villain. Besides, Batman must use the powers of his list in order (possibly skipping some powers) and have to beat all the villains in the same order of the list. Like everybody else, Batman doesn't have unlimited energy . . . he can only spend maximum E calories per day.

Given a list of superpowers and a list of villains, is it possible that Batman beats them all without spending more than E calories?

Input

The input consists of several test cases. Each test case is represented as follows:

- A line with three integers P, V and E ($0 \leq P, V \leq 1000, 0 \leq E \leq 10^8$), representing (respectively) the number of superpowers, the number of villains and the maximum amount of calories that Batman can spend per day.
- P lines, one per superpower, containing the following information separated by spaces:
  - An alphanumeric string s with the name of the superpower (case sensitive, without spaces).
  - An integer a ($0 \leq a \leq 10000$) specifying the attack factor of the superpower.
  - An integer c ($0 \leq c \leq 100000$) specifying the energy consumption in calories of the superpower.
- V lines, one per villain, containing the following information separated by spaces:
  - An alphanumeric string s with the alias of the villain (case sensitive, without spaces).
  - An integer d ($0 \leq d \leq 10000$) specifying the defense factor of the villain.
  - A list with the names of the superpowers that can affect the villain, separated with commas. The list will not contain repeated names.

The end of the input is specified by a line with the string `'0 0 0'`. Suppose that the superpowers have distinct names and that the villains have distinct aliases. The maximum length of a name or alias is 100 characters and the minimum length of a name or alias is 1 character.

The input must be read from standard input.

Output

For each test case, print the line `'Yes'` if it's possible for Batman to beat all the villains spending E or less calories. Otherwise, print the line `'No'`.

The output must be written to standard output.

Sample Input	**Sample Output**
4 3 1200 A 8000 500 B 7000 600 C 9000 400 D 5000 300 Joker 6000 B,A Penguin 8000 B,C,D TwoFace 5000 D,A,B,C 4 3 1000 A 8000 500 B 7000 600 C 9000 400 D 5000 300 Joker 6000 B,A Penguin 8000 B,C,D TwoFace 5000 D,A,B,C 0 0 0	Yes No

C - Simple Minded Hashing

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

All of you know a bit or two about hashing. It involves mapping an element into a numerical value using some mathematical function. In this problem we will consider a very 'simple minded hashing'. It involves assigning numerical value to the alphabets and summing these values of the characters.

For example, the string "acm" is mapped to 1 + 3 + 13 = 17. Unfortunately, this method does not give one-to-one mapping. The string "adl" also maps to 17(1 + 4 + 12). This is called collision.

In this problem you will have to find the number of strings of length L, which maps to an integer S, using the above hash function. You have to consider strings that have only lowercase letters in strictly ascending order.

Suppose $L = 3$ and $S = 10$, there are 4 such strings.

1. abg
2. acf
3. ade
4. bce

"agb" also produces 10 but the letters are not strictly in ascending order. "bh" also produces 10 but it has 2 letters.

Input

There will be several cases. Each case consists of 2 integers L and S ($0 < L, S < 10000$). Input is terminated with 2 zeros.

The input must be read from standard input.

Output

For each case, output `'Case#:'` where # is replaced by case number. Then output the result. Follow the sample for exact format. The result will fit in 32 signed integers.

The output must be written to standard output.

Sample Input	**Sample Output**

`3 10`
`2 3`
`0 0`
	`Case 1: 4`
	`Case 2: 1`

D - Mega Man Missions

Source file name: `megaman.py`
Time limit: 3 seconds

Mega Man is off to save the world again. His objective is to kill the Robots created by Dr. Wily whose motive is to conquer the world. In each mission, he will try to destroy a particular Robot. Initially, Mega Man is equipped with a weapon, called the "Mega Buster" which can be used to destroy the Robots. Unfortunately, it may happen that his weapon is not capable of taking down every Robot. However, to his fortune, he is capable of using the weapons from Robots which he has *completely destroyed* and these weapons maybe able to take down Robots which he otherwise cannot with his own weapon. Note that, each of these enemy Robots carry exactly one weapon themselves for fighting Mega Man. He is able to take down the Robots in any order as long as he has at least one weapon capable of destroying the Robot at a particular mission. In this problem, given the information about the Robots and their weapons, you will have to determine the number of ways Mega Man can complete his objective of destroying all the Robots.

Input

Input starts with an integer T, the number of test cases.

Each test case starts with an integer N ($1 \leq N \leq 16$). Here N denotes the number of Robots to be destroyed (each Robot is numbered from 1 to N). This line is followed by $N + 1$ lines, each containing N characters. Each character will either be `'1'` or `'0'`. These lines represent a $(N + 1) \times N$ matrix. The rows are numbered from 0 to N while the columns are numbered from 1 to N. Row 0 represents the information about the "Mega Buster". The j-th character of Row 0 will be `'1'` if the "Mega Buster" can destroy the j-th Robot. For the remaining N rows, the j-th character of i-th row will be `'1'` if the weapon of i-th Robot can destroy the j-th Robot. Note that, a Robot's weapon could be used to destroy the Robot itself, but this will have no impact as the Robot must be destroyed anyway for its weapon to be acquired.

The input must be read from standard input.

Output

For each case of input, there will be one line of output. It will first contain the case number followed by the number of ways Mega Man can complete his objective. Look at the sample output for exact format.

The output must be written to standard output.

Sample Input	**Sample Output**
3 1 1 1 2 11 01 10 3 110 011 100 000	Case 1: 1 Case 2: 2 Case 3: 3

E - Unique World

Source file name: `unique.py`
Time limit: 10 seconds

Almost everything is unique in Unique world. The unique creature, unika lives in this world. Each unika has its own unique house. All the houses are connected by roads in such a way so that there is exactly one (unique !!) path from one house to another one. More importantly the cost to traverse each road is unique. In a recent observation, Xenique (President of Unique world) found that the costs of all possible paths are not unique and this should not happen in the Unique World. So, he decided that the cost of each path would be unique. To make this, he also decided that unika can traverse same road more than once while going from one house to another one except the last road on the path. But soon or later he was informed that too many traffic jam is being created due to this rule. Xenique called a meeting on this issue.

A young unika, Prognique suggested that they should use minimum number of roads while visiting from one place to another one. He also told that they should use only those roads, which are necessary for the path. And only a computer program can help in this case.

Input

The first line of the input file contains a single integer N ($0 < N \leq 20$) that denotes the number of inputs. Each data set starts with two positive integers n ($n < 51$), which denotes the number of unika and m, the number of roads in the unique world. Each of next m lines contains three positive integers, the *id* of unika (Each unika has a unique *id* between 1 to n) id_1, id_2 and c. There is a road of traversing cost c connecting the house of these two unika. Next line contains another integer k ($1 \leq k \leq 20$), the number of queries. Each of next k lines contains three integers the *id* of the unika of the source and destination houses and the required cost (between 1 and 100000) of this path. Input may contain blank lines.

The input must be read from standard input.

Output

For each data set, if it is possible to traverse the path using the given cost, print `'Yes'` and the minimum number of roads. Otherwise just print `'No'`. Put a blank line between two consecutive sets of inputs.

The output must be written to standard output.

Sample Input	**Sample Output**
1 5 4 1 2 2 1 3 3 1 4 4 1 5 5 5 1 2 2 2 3 5 2 3 6 4 5 10 2 4 18	Yes 1 Yes 2 No No Yes 8