ADA - Análisis y Diseño de Algoritmos, 2020-2
Tarea 3: Semanas 6 y 7

Para entregar el viernes 25/domingo 27 de septiembre de 2020

Problemas conceptuales a las 23:59 (25 de septiembre) en el Departamento de Electrónica y Ciencias de la Computación

Problemas prácticos a las 23:59 (27 de septiembre) 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

16.1-1, 16.1-2, 16.1-3, 16.1-4, 16.1-5 (página 422), 16.2-1, 16.2-3, 16.2-5, 16.2-7 (páginas 427 y 428).

Problemas conceptuales

1. Problema 16-2: *Scheduling to minimize average completion time* (Cormen et al., página 447).
2. Ejercicio 4.9: *Minimum-bottleneck spanning tree* (Kleinberg & Tardos, página 192).
3. Ejercicio 4.1: *Greedy Schedule* (Erickson, página 176).

Problemas prácticos

Hay seis (uno es bono) problemas prácticos cuyos enunciados aparecen a partir de la siguiente página.

A - Alien DNA

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

Alien DNA is more complicated than human DNA. For example, rather than viewing an alien DNA strand as a sequence of base pairs, it can be viewed as a sequence of base sets. A base set is, simply put, a string of lowercase letters where each letter represents a different base. Note that human DNA is also a sequence of base sets, except a base set is simply a pairing of a with t or c with g.

Since alien DNA is more complicated than human DNA, the algorithms used in alien bioinformatics are also much more complicated. However, you have only been working at AlienBioTechNextGenCorp for three days and are not yet trusted enough to handle the complicated tasks.

Your first job is simple. You must develop software to cut a strand of alien DNA into as few segments as possible. The only constraint is that base sets of a given segment must share at least one common base.

Input

Input begins with an integer $t \geq 0$, the number of test cases to be processed. Each test case begins with a single value n ($1 \leq n \leq 10\,000$) representing the length of the alien DNA strand. Each of the n following lines consists of a single string of lowercase letters representing a base set. Each string will contain at least one character, and no repeated characters. The base sets are given in the input in the same order they appear on the DNA strand.

The input must be read from standard input.

Output

For each test case, output the minimum number of cuts required to partition the strand into segments that share a common base.

The output must be written to standard output.

Sample Input	**Sample Output**
2 5 as sd df fg gh 3 plum orange plum	2 2

B - Cinema-cola

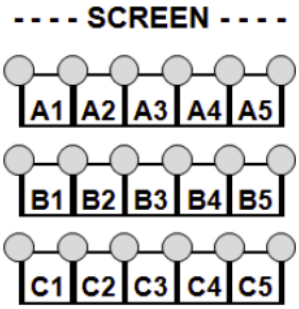
Source file name: `cinema.py`
Time limit: 1 second

A group of Z friends is going to the movies. They have already reserved some specific locations in the theater. As a kind of ritual, every friend likes to drink something while seeing the film.

Locations at the theater are disposed in a rectangular array of R rows and C columns. Rows are named sequentially with English capital letters ('`A`', '`B`', '`C`', ...). Columns are named with integers $(1, 2, 3, \ldots, C)$. Then, locations are identified by a string formed by the corresponding row letter and column number of the location.

Locations at the theater are chairs with plastic supports at both arms, to put a drink on each one. Supports are shared by neighbor locations but, of course, it is expected that at most one person uses a support to put his/her drink, but there is no etiquette rule that forces someone to use the right or the left support of his/her chair.

As a matter of example, the next figure illustrates a theater with 3 rows and 5 columns. Locations are depicted with rectangular forms, and supports with circles at both sides of them.



The group of friends goes to occupy the reserved locations. Before them, some persons came and occupied some of the locations (different from those of the group) and everyone used one support for a drink. Is it possible that the group of friends sit at their locations warranting that everyone in the group can put his/her drink on a support that corresponds to his/her location? Clearly, sometimes the spectators that came before the group may have used the supports in such a way that not everyone in the group may use a support for his/her drink.

Input

There are several cases to consider. Each case begins with a line with two integer numbers R and C, indicating the number of rows and columns in the theater ($1 \leq R \leq 26$, $2 \leq C \leq 99$). The next line contains an integer number P indicating the number of persons that came to the theater before than the group ($0 \leq P \leq R \cdot C - 1$). Each one of the next P lines contains two text strings; the first one corresponds to the location id (one letter, one integer numeral) and, the second one is an one-character string with a sign '`-`' or a sign '`+`' denoting the fact that the person is using the left or the right support of his/her location, respectively. The next line contains an integer number Z indicating the number of persons in the group of friends ($1 \leq Z \leq R \cdot C - P$). Finally, each one of the following Z lines contains the locations' ids of the reserved seats for the group of friends. In each case, you may suppose that the given identifiers are distinct and that every drink was put in an empty support before the friends came.

Each line in a case description has no leading spaces and items on a line are separated by exactly one space. There are no spaces at the end of any line. Input ends with a line with two '`0`' values.

The input must be read from standard input.

Output

For each case output a line with one of the texts '`YES`' or '`NO`' depending on the fact that the group of friends can occupy their seats and use some of the supports to place their drinks with the constraints imposed by the already occupied seats and supports.

The output must be written to standard output.

Sample Input	**Sample Output**
3 5 3 A3 - B1 - B4 - 4 A1 A2 B2 B3 3 5 3 A3 - B1 + B4 - 4 A1 A2 B2 B3 0 0	YES NO

C - Elephants

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

"N elephants went out to play on a spider web one day". Having a set of M elephants, each one with a weight w_i where $1 \leq i \leq M$, and knowing the maximum weight that the spider web supports, what is the largest number of elephants that you can put in the spider web without breaking it?

Input

The first line of input contains a non negative integer meaning the number of test cases. Each case starts with a line with two integers M and W, the number of elephants and the maximum weight that the spider web supports ($1 \leq M \leq 10^5$ and $1 \leq W \leq 10^8$). The next line contains M numbers w_i representing the weight of each elephant ($1 \leq w_i \leq 10000$).

The input must be read from standard input.

Output

Print a single line per test case with the largest number of elephants that you can put in the spider web without breaking it.

The output must be written to standard output.

Sample Input	**Sample Output**
3 5 14 10 15 16 17 18 4 20 1 2 3 4 5 22 9 1 8 7 7	1 4 3

D - Gas Stations

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

G gas stations are authorized to operate over a road of length L. Each gas station is able to sell fuel over a specific area of influence, defined as the closed interval $[x - r, x + r]$, where x is the station's *location* on the road ($0 \leq x \leq L$) and r is its *radius of coverage* ($0 < r \leq L$). The points covered by a gas station are those within its radius of coverage.

It is clear that the areas of influence may interfere, causing disputes among the corresponding gas stations. It seems to be better to close some stations, trying to minimize such interferences without reducing the service availability along the road.

The owners have agreed to close some gas stations in order to avoid as many disputes as possible. You have been hired to write a program to determine the maximum number of gas stations that may be closed, so that every point on the road is in the area of influence of some remaining station. By the way, if some point on the road is not covered by any gas station, you must acknowledge the situation and inform about it.

Input

The input consists of several test cases. The first line of each test case contains two integer numbers L and G (separated by a blank), representing the length of the road and the number of gas stations, respectively ($1 \leq L \leq 10^8$, $1 \leq G \leq 10^4$). Each one of the next G lines contains two integer numbers x_i and r_i (separated by a blank) where x_i is the location and r_i is the radius of coverage of the i-th gas station ($0 \leq x_i \leq L$, $0 < r_i \leq L$). The last test case is followed by a line containing two zeros.

The input must be read from standard input.

Output

For each test case, print a line with the maximum number of gas stations that can be eliminated, so that every point on the road belongs to the area of influence of some not closed station. If some point on the road is not covered by any of the initial G gas stations, print '`-1`' as the answer for such a case

The output must be written to standard output.

Sample Input	**Sample Output**
40 3 5 5 20 10 40 10 40 5 5 5 11 8 20 10 30 3 40 10 40 5 0 10 10 10 20 10 30 10 40 10 40 3 10 10 18 10 25 10 40 3 10 10 18 10 25 15 0 0	0 2 3 -1 1

E - Fill

Source file name: `fill.py`
Time limit: x seconds

There are three jugs with a volume of a, b and c liters. (a, b, and c are positive integers not greater than 200). The first and the second jug are initially empty, while the third is completely filled with water. It is allowed to pour water from one jug into another until either the first one is empty or the second one is full. This operation can be performed zero, one or more times.

You are to write a program that computes the least total amount of water that needs to be poured; so that at least one of the jugs contains exactly d liters of water (d is a positive integer not greater than 200). If it is not possible to measure d liters this way your program should find a smaller amount of water $d' < d$ which is closest to d and for which d' liters could be produced. When d' is found, your program should compute the least total amount of poured water needed to produce d' liters in at least one of the jugs.

Input

The first line of input contains the number of test cases. In the next T lines, T test cases follow. Each test case is given in one line of input containing four space separated integers — a, b, c and d.

The input must be read from standard input.

Output

The output consists of two integers separated by a single space. The first integer equals the least total amount (the sum of all waters you pour from one jug to another) of poured water. The second integer equals d, if d liters of water could be produced by such transformations, or equals the closest smaller value d' that your program has found.

The output must be written to standard output.

Sample Input	**Sample Output**
2 2 3 4 2 96 97 199 62	2 2 9859 62

F - Magic Car

Source file name: `magic.py`
Time limit: x seconds

ACM (Association of Car Modernization) has recently developed a new car, "MAGIC CAR". It uses solar energy. The car has some interesting characteristics:

- It uses up constant amount of energy at the start for any initial speed.
- It also uses up constant energy when stops.
- If it changes speed to a value which is less than already achieved least speed or greater than already achieved most speed, it uses up some energy. In both cases the energy is equal to the absolute difference of the current speed and previously achieved least or most speed.
- The loss of energy does not depend on the distance the car covered (Really magic!!!).

Mr. Oberoy has such a magic car. So far he has used the car quite intelligently so that minimum energy is used up. This was easy for him as he could take any speed on any road. But recently, TCD (Transport Control Department) has decided that there will be a fixed speed for each road in the city and everybody must maintain the speed. Mr. Oberoy is in problem now. Can you help him so that he can optimally use the car?

Input

Each dataset starts with two positive integer, N ($2 \leq N \leq 200$) denoting the number of junctions and M ($1 \leq M \leq 1000$) denoting the number of roads in Mr. Oberoy's city. Each junction is identified by a unique integer from 1 to N. In next few lines there will be road descriptions. A road is described by three positive integers which are start, end junctions and the fixed speed of that road. There may be more than one roads between two junctions. Roads are bidirectional. In the next line there will be two positive integers which are the used up energy during start and stop of the magic car. Next line will contain an integer K ($1 \leq K \leq 5$) indicating the number of queries. Each of following K lines will contain two integers, the source and destination junction of Mr. Oberoy. Source and destination will not be same. Input is terminated by EOF.

The input must be read from standard input.

Output

For each dataset print the minimum possible used up energy for each query of Mr. Oberoy. It is guranteed that the destination is always reachable from source.

The output must be written to standard output.

Sample Input	**Sample Output**
4 4 1 2 2 2 3 4 1 4 1 3 4 2 5 5 2 1 3 1 2	11 10