Árboles y Grafos, 2025-2

Para entregar el miércoles 22 de octubre de 2025
A las 23:59 en la arena de programación

Instrucciones para la entrega

- Para esta tarea y todas las tareas futuras en la arena de programación, la entrega de soluciones es *individual*. Por favor escriba claramente su nombre, código de estudiante y sección en cada archivo de código (a modo de comentario). Adicionalmente, cite cualquier fuente de información que utilizó. Los códigos fuente que suba a la arena de programación deben der de su completa autoría.
- En cada problema debe leer los datos de entrada de la forma en la que se indica en el enunciado y debe imprimir los resultados con el formato allí indicado. No debe agregar mensajes ni agregar o eliminar datos en el proceso de lectura. La omisión de esta indicación puede generar que su programa no sea aceptado en la arena de programación.
- Puede resolver los ejercicios en C/C++ y Python. Sin embargo, deben haber por los menos soluciones a dos problemas en cada lenguaje.
- Debe enviar sus soluciones a través de la arena. Antes de subir sus soluciones asegurese de realizar pruebas con los casos de pruebas proporcionados para verificar que el programa es correcto, que finaliza y no se quede en un ciclo infinito.
- El primer criterío de evaluación será la aceptación del problema en la arena cumpliendo los requisitos indicados en los enunciados de los ejercicios y en este documento. El segundo criterio de evaluación será la complejidad computacional de la solución y el uso de los temas vistos en clase. Es necesario incluir en la cabecera del archivo comentarios que expliquen la complejidad de la solución del problema para cada caso.

  En adición a lo anterior, para efectos de la calificación se tendrán en cuenta aspectos de estilo como no usar `break` ni `continue` y que las funciones deben tener únicamente una instrucción `return` que debe estar en la última línea.

Problemas prácticos

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

A - Problem A

Source file name: `digits.cpp, digits.py`
Time limit: 20 seconds

Digits like to dance. One day, $1, 2, 3, 4, 5, 6, 7$ and 8 stand in a line to have a wonderful party. Each time, a male digit can ask a female digit to dance with him, or a female digit can ask a male digit to dance with her, as long as their sum is a prime. Before every dance, exactly one digit goes to who he/she wants to dance with - either to its immediate left or immediate right.

For simplicity, we denote a male digit x by itself x, and denote a female digit x by $-x$. Suppose the digits are in order $\{1, 2, 4, 5, 6, -7, -3, 8\}$. If -3 wants to dance with 4, she must go either to 4's left, resulting $\{1, 2, -3, 4, 5, 6, -7, 8\}$ or his right, resulting $\{1, 2, 4, -3, 5, 6, -7, 8\}$. Note that -3 cannot dance with 5, since their sum $3 + 5 = 8$ is not a prime; 2 cannot dance with 5, since they're both male.

Given the initial ordering of the digits, find the minimal number of dances needed for them to sort in increasing order (ignoring signs of course).

Input

The input consists of multiple test cases. Each case contains exactly 8 integers in a single line. The absolute values of these integers form a permutation of $\{1, 2, 3, 4, 5, 6, 7, 8\}$. The last case is followed by a single zero, which should not be processed.

The input must be read from standard input.

Output

For each test case, print the case number and the minimal number of dances needed. If they can never be sorted in increasing order, print `'-1'`.

The output must be written to standard output.

Sample Input

```
1 2 4 5 6 -7 -3 8
1 2 3 4 5 6 7 8
1 2 3 5 -4 6 7 8
1 2 3 5 4 6 7 8
2 -8 -4 5 6 7 3 -1
0
```

Sample Output

```
Case 1: 1
Case 2: 0
Case 3: 1
Case 4: -1
Case 5: 3
```

B - Problem B

Source file name: `obsession.cpp, obsession.py`
Time limit: 1 second

Patricia is an excellent software developer, but, as every brilliant person, she has some strange quirks. One of those is that everything she does has to be in even quantities. Most often that quirk does not affect her, even though it may seem strange to others. Some examples: every day she has to eat an even number of meals; during breakfast, she drinks two cups of coffee, eats two toasts and two slices of cheese; when she goes to the cinema she buys two tickets (fortunately she always has a friend that goes with her); she takes two baths per day (or four, our six . . .).

Some other times, however, that quirk makes the life of Patricia more difficult. For example, no one wants to travel by car with her because if she has to pay toll, the number of tolls she pays has to be an even number.

Patricia lives in a country where all roads are two-way and have exactly one toll each. She needs to visit a client in a different city, and wants to calculate the minimum total value of tolls she has to pay to go from her city to the client's city, obeying her strange quirk that she has to pay an even number of tolls.

Input

The input consists of several test cases. The first line of a test case contains two integers C and V, the total number of cities and the number of roads ($2 \leq C \leq 10^4$ and $0 \leq V \leq 50000$). The cities are identified by integer numbers from 1 to C. Each road links two different cities, and there is at most one road between each pair of cities. Each of the next V lines contains three integers C_1, C_2 and G, indicating that the toll value of the road linking cities C_1 and C_2 is G ($1 \leq C1, C2 \leq C$ and $1 \leq G \leq 10^4$). Patricia is currently in city 1 and the client's city is C.

The input must be read from standard input.

Output

For each test case in the input your program must output exactly one line, containing exactly one integer, the minimum toll value for Patricia to go from city 1 to city C, paying an even number of tolls, or, if that is not possible, the value `'-1'`.

The output must be written to standard output.

Sample Input	Sample Output

4 4
1 2 2
2 3 1
2 4 10
3 4 6
5 6
1 2 3
2 3 5
3 5 2
5 1 8
2 4 1
4 5 4
	12
	-1

C - Problem C

Source file name: `parabox.cpp, parabox.py`
Time limit: 1 second

Professor Farnsworth, a renowned scientist that lives in year 3000 working at *Planet Express Inc.*, performed a failed experiment that nearly killed him. As a sub-product, some strange boxes were created. Farnsworth gave one of the boxes to Leela, who accidentally discovered that it leads to a parallel universe. After that, the *Planet Express* crew traveled to the new discovered parallel universe using the box, meeting their corresponding parallel copies, including a parallel Professor Farnsworth who also created some boxes.

Simultaneously, some parallel copies of the Professor created similar boxes in some existing parallel universes. As a result, some universes, including the original one, were endowed with a (possibly empty) collection of boxes leading to other parallel universes. However, the boxes have a bug: besides allowing travels among different parallel universes, they allow for time travels. So, a particular box leads to a distinct parallel universe possibly allowing a voyager to gain or lose a certain number of time units.

More precisely, given two distinct universes A and B, and a non-negative integer number t, a (A, B)-box with time displacement t is an object designed to travel between the two universes that can be used directly (traveling from A to B) or reversely (traveling from B to A). A such box exists in both universes, allowing travels among both universes. A voyager that uses the (A, B)-box directly can travel from universe A to universe B landing t time units in the future. On the other hand, a voyager that uses the (A, B)-box reversely can travel from universe B to universe A landing t time units in the past. Box building requires so much energy that there may be built at most one box to travel between a given pair of different universes.

A *circuit* is defined as a non-empty sequence of parallel universes $\langle s_1, s_2, \ldots, s_m\rangle$ such that:

- The first and the last universe in the sequence are the same (i.e., $s_1 = s_m$).
- For every k ($1 \leq k < m$) there is a (s_k, s_{k+1})-box or a (s_{k+1}, s_k)-box to travel (directly or reversely) from universe s_k to universe s_{k+1}.

The possible existence of circuits is very interesting. Using the corresponding boxes of a circuit, a voyager may experiment real time travels. Professor Farnsworth wants to know if there is a circuit that starts in the original universe and allows travels to the past, constituting a phenomenon known as the Farnsworth Parabox. For example, imagine that there are three universes, A, B and C, and that there exist the following boxes: a (A, B)-box with time displacement 3, a (A, C)-box with time displacement 2, and a (B, C)-box with time displacement 4. Clearly, the sequence $\langle A, B, C, A\rangle$ is a circuit, that allows to travel five time units in the future, starting and ending at universe A.

The original Farnsworth Professor, who lives in the original universe, wants to know if the Farnsworth Parabox is true or not. Can you help him?

Input

There are several cases to solve. Each case begins with a line with two integer numbers N and B, indicating the number of parallel universes (including the original) and the number of existing boxes, respectively ($2 \leq N \leq 100, 1 \leq B \leq N \cdot (N-1)/2$). The distinct universes are identified uniquely with the numbers $1, 2, \ldots, N$, where the original universe is the number 1. Each one of the next B lines contains three integer numbers i, j and t, describing a (i, j)-box to travel between the universe i and the universe j with time displacement t ($1 \leq i \leq N, 1 \leq j \leq N, i \neq j, 0 \leq t \leq 100$). The input ends with a line with two `0` values.

The input must be read from standard input.

Output

For each test case output one line with the letter `'Y'` if the Farnsworth Parabox is true; or with the letter `'N'`, otherwise.

The output must be written to standard output.

Sample Input

```
2 1
2 1 1
3 3
1 2 3
1 3 2
2 3 4
4 4
1 2 2
3 2 2
3 4 2
1 4 2
0 0
```

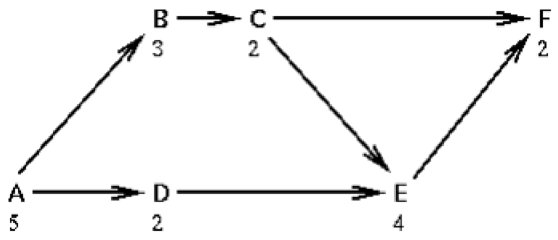
Sample Output

```
N
Y
N
```

D - Problem D

Source file name: `project.cpp, project.py`
Time limit: 1 second

A project management technique called Pert involves breaking a large project into a number of tasks, estimating the time required to perform each task, and determining which tasks can not be started until others have been completed. The project is then summarized in chart form. For example, the chart (which corresponds to the sample input below)



indicates that tasks *A*, *B*, *C*, *D*, *E* and *F* each take 5, 3, 2, 2, 4, and 2 days respectively, that task *E* cannot complete until *C* and *D* are both completed, but that *D* can be performed in parallel with *B* and *C*.

Write a program that accepts a Pert chart and computes the amount of time required to complete a project.

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. Input will be from 1 to 27 lines, each corresponding to a different task. Each line will contain:

1. A single upper case letter serving as the name of a task. On the final line of input, this will be blank and the rest of that line is ignored.
2. An integer indicating the number of days required to complete that task.
3. 0-26 additional uppercase letters, each indicating another task that must complete before this one can begin.

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.

The output is a single integer indicating the amount of time that will pass before all tasks can complete.

The output must be written to standard output.

Sample Input

```
2

A 5
B 3 A
D 2 A
C 2 B
F 2 CE
E 4 DC

A 5
B 3 A
D 2 A
C 2 B
F 2 CE
E 4 DC
```

Sample Output

```
16

16
```

E - Problem E

Source file name: `zlatan.cpp, zlatan.py`
Time limit: 3 seconds

In the AGRAdable and AGRAnd Empire of Zlatan there are train stations connected by one-way tracks. Each track has an associated non-negative cost, and the total cost of a trip equals the sum of the costs of the tracks used. A city is defined as a group of stations such that, regarding mobility, it is guaranteed that from any station in the group it is possible to reach all the others in the same group. Cities are connected to each other by a single one-way track. Zlatanians feel deeply AGRAvated because, when traveling between different cities, it is not possible to go and return using the train.

Kenny lives in one of the stations and, since he has very limited resources, he never leaves his city. To minimize his expenses, he always chooses the cheapest possible routes when traveling within his city. For that reason, Kenny would like to know in advance the minimum travel cost between every pair of stations in his city. He is not interested in stations outside of it, since for him it is not AGRAdable to think about things he will never use.

Input

The input consists of multiple test cases. Each test case begins with two integers N ($2 \leq Nleq2,000$) and M ($0 \leq N \cdot (N-1)/2$) representing the number of train stations and the number of one-way railways, respectively. Each of the next M lines contains three integers u, v, and c ($1 \leq u, v \leq N, 1 \leq c \leq 500$), indicating that there is a one-way railway from station u to station v with a cost of c. Finally, a line with one integer k ($1 \leq k \leq N$) is given, representing the index of the station where Kenny's house is located.

The input ends with a line containing `0 0`.

The input must be read from standard input.

Output

For each test case, consider only the stations that belong to the same city as Kenny's station. For every ordered pair of such stations (u, v), print a line with three integers:

```
u v c
```

where c is the minimum total cost of traveling from station u to station v within the same city. Since within a city it is always possible to travel between any two stations, all pairs will have a valid finite cost. The triplets must be printed in **lexicographic order**. Print a blank line between test cases.

The output must be written to standard output.

Sample Input

```
12 14
1 3 2
3 2 1
2 6 5
6 7 1
7 1 3
10 4 2
4 9 1
9 10 4
5 12 6
12 8 2
8 11 1
11 5 3
3 10 8
4 8 7
3
0 0
```

Sample Output

```
1 2 3
1 3 2
1 6 8
1 7 9
2 1 9
2 3 11
2 6 5
2 7 6
3 1 10
3 2 1
3 6 6
3 7 7
6 1 4
6 2 7
6 3 6
6 7 1
7 1 3
7 2 6
7 3 5
7 6 11
```