Trainingsplan

Arthur hat beschlossen etwas für seine Figur zu tun und ins Fitnessstudio zu gehen. Das Fitnessstudio bietet insgesamt n verschiedene Geräte zum Trainieren an. Diese sind von 1 bis n durchnummeriert.

Einer seiner Freunde hat ihm sogar einen Trainingsplan zusammengestellt, welcher aus m Schritten besteht. Die Schritte sind genauso von 1 bis m durchnummeriert. Wenn er nun also das Fitnessstudio besucht, arbeitet er diese m Schritte der Reihe nach ab.

Jeder Schritt ist entweder eine *Übungsphase* oder eine *Wiederholungsphase*:

- `Übungsphase:` Arthur trainiert an jedem Gerät im Intervall $[l, r]$. Das heißt er übt mit den Geräten Nummer $l,\ l+1,\ \ldots\ r-1,\ r$ jeweils *einmal*.
- `Wiederholungsphase:` Sein Freund weiß, dass eine einzige Wiederholung an den Geräten wohl nicht genügen wird. Deshalb gilt es in dieser Phase alle Schritte im Intervall $[l, r]$ noch einmal zu wiederholen. Diese Schritte liegen natürlich *vor* dem aktuellen Schritt, da sie ja wiederholt werden. Sollte bei diesen eine weitere Wiederholungsphase dabei sein, so wiederholt Arthur diese auch.

All dieses Training macht Arthur schwer zu schaffen. Er ist der Meinung, dass sein Freund es mit dem Trainingsplan etwas zu gut gemeint hat. Leider hat er aber mit all den Wiederholungen schon den Überblick verloren. Kannst du Arthur helfen und herausfinden, wie oft er an jedem Gerät trainiert hat? Diese Werte können potentiell (wie er schon vermutet hat) ziemlich groß werden. Gib sie deshalb *modulo* $10^9 + 7$ (1000000007) aus.

Eingabe

Die erste Zeile des Inputs enthält zwei Zahlen n und m, die Anzahl an Geräte und die Anzahl an Schritte des Trainingsplans. Die nächsten m Zeilen enthalten jeweils drei Zahlen $t\ l\ r$, die die einzelnen Schritte beschreiben. $t = 1$ bedeutet, dass es sich um eine Übungsphase handelt, $t = 2$ hingegen kennzeichnet eine Wiederholungsphase. $l\ r$ beschreibt das dazugehörige Intervall.

Ausgabe

Gib n Zahlen aus, die Anzahl an Wiederholungen, die er an jedem Gerät macht - modulo $10^9 + 7$ (1000000007).

Beispiel

Eingabe	Ausgabe
5 5 1 1 2 1 4 5 2 1 2 1 3 4 2 3 4	3 3 2 5 3

Im ersten Schritt trainiert Arthur an Gerät 1 und 2 einmal, im zweiten mit Gerät 4 und 5. Der dritte Schritt wiederholt die ersten Beiden. Somit ist zu diesem Zeitpunkt jedes Gerät, außer das Dritte, zweimal benutzt worden. Im vierten Schritt trainiert er an Gerät 3 und 4. Zum Schluss wird die erste Wiederholungsphase erneut durchgeführt, sowie nochmal an Gerät 3 und 4 trainiert.

Subtasks

Allgemein gilt:

- $1 \leq n, m \leq 10^5$
- $1 \leq t \leq 2$
- $1 \leq l \leq r \leq n$ bei Übungsphasen
- $1 \leq l \leq r \leq m$ bei Wiederholungsphasen
- r ist bei Wiederholungsphasen *kleiner* als die Nummer des aktuellen Schrittes. Er wiederholt also tatsächlich nur Schritte, die er schon gemacht hat. Somit ist die erste Phase seines Trainingsplanes auch immer eine Übungsphase.

Subtask 1 (4 Punkte): $n, m \leq 1000$, $t = 1$ (nur Übungsphasen)

Subtask 2 (26 Punkte): $t = 1$

Subtask 3 (5 Punkte): $n, m \leq 10$

Subtask 4 (25 Punkte): $n, m \leq 1000$

Subtask 5 (40 Punkte): Keine Einschränkungen

Beschränkungen

Zeitlimit: 1 s **Speicherlimit:** 256 MB

Hinweis Modulo

Wenn Werte zu groß werden können, dann muss man diese bei Programmierwettbewerben oft modulo (Divisionsrest, der % Operator in C++/Java) einer großen (Prim-)Zahl ausgeben. Dies soll die Aufgabe nicht schwieriger machen. Ganz im Gegenteil, der Grund dafür ist, dass man so beim Berechnen keine `BigInteger` (Datentypen mit mehr als 64 Bit) benötigt. Es gilt:

$$(a + b) \% c = ((a \% c) + (b \% c)) \% c \tag{1}$$

$(10 + 5) \% 7$ liefert also das selbe Ergebnis wie $((10 \% 7) + (5 \% 7)) \% 7 = (3 + 5) \% 7 = 1$.

Auch multiplizieren ist funktioniert so:

$$(a \cdot b) \% c = ((a \% c) \cdot (b \% c)) \% c \tag{2}$$

Ähnlich beim Subtrahieren:

$$(a - b) \% c = ((a \% c) - (b \% c) + c) \% c \tag{3}$$

Das $+ c$ ist dazu da, dass die Summe auf jeden Fall wieder Positiv wird.

Dividieren ist etwas komplizierter und wird zum Lösen dieser Aufgabe nicht zwingend benötigt.

Diese Gleichungen erlauben es, dass man schon bei den Zwischenschritten modulo rechnet, und so nie einen Overflow produziert.

Zum Beispiel: *Berechne das Produkt aller Elemente eines Arrays a, mit Länge n - modulo* 10^6. *Es gilt:* $1 \leq n \leq 1000$, $1 \leq a_i \leq 1000$

Ein 32-Bit Datentyp wie (zumindest auf den meisten Systemen) `int` kann Werte bis zu $2^{31} - 1 = 2147483647$, also etwas über zwei Milliarden ($2 \cdot 10^9$), speichern. Wenn wir in diesem Beispiel die Werte einfach multiplizieren, dann könnte dieses Produkt bis zu 1000^{1000} - also weit über dem `int` Range - groß werden. Wenn wir die Zahlen also naive multiplizieren und dann modulo Rechnen, dann entsteht ein Overflow. Deshalb kommt der Modulo ins Spiel:

Durch Gleichung (2) wissen wir, dass wir einfach zwischendurch modulo rechnen dürfen, und das Ergebnis dadurch nicht verändert wird:

```
int product = 1;
for (int i = 0; i < n; i++)
    product = (product * a[i]) % 1000000;
```

Wenn wir modulo x rechnen, dann ist das Ergebnis zwischen 0 und $x - 1$ (es ist ja immerhin der Rest). Somit kann das Produkt (product * a[i]) maximal $(1000000 - 1) \cdot 1000 < 10^9$ groß werden, was im `int` Range liegt. Deshalb können wir das Produkt aller Zahlen - modulo 10^6 - ausrechnen, ohne je mehr als einen 32-Bit Datentypen zu benötigen.